ALPHA MICROSYSTEMS
RIGHT. FROM THE START.

ISAM System User's Guide

DWM-00100-06, Rev. C01

© 1977, 1979, 1981, 1982, 1983, 1984, 1992 Alpha Microsystems

REVISIONS INCORPORATED	
REVISION	DATE
A00	December 6, 1977
A01	April 1979
A02	April 30, 1981
A03	October 31, 1981
B00	June 1, 1982
C00	March 18, 1983
C01	May 21, 1984

ISAM System User's Guide
To re-order this document, request part number DWM-00100-06.

This document applies to AMOS/T Versions 5.0 and later
and AMOS Versions 1.1 and later.

The information contained in this manual is believed to be accurate and reliable. However, no responsibility for the accuracy, completeness or use of this information is assumed by Alpha Microsystems.

This document may contain references to products covered under U.S. Patent Number 4,530,048.

The following are registered trademarks of Alpha Microsystems, Santa Ana, CA 92799:

AMIGOS	AMOS	Alpha Micro	AlphaACCOUNTING
AlphaBASIC	AlphaCALC	AlphaCOBOL	AlphaFORTRAN 77
AlphaLAN	AlphaLEDGER	AlphaMAIL	AlphaMATE
AlphaNET	AlphaPASCAL	AlphaRJE	AlphaWRITE
CASELODE	OmniBASIC	VER-A-TEL	VIDEOTRAX

The following are trademarks of Alpha Microsystems, Santa Ana, CA 92799:

AlphaBASIC PLUS	AlphaVUE	AM-PC	AMTEC
DART	ESP	MULTI	

ALPHA MICROSYSTEMS
3501 Sunflower
P.O. Box 25059
Santa Ana, CA 92799

Table of Contents

CHAPTER 1 ABOUT THIS BOOK

1.1 COMPATIBILITY NOTES 1-2
1.2 GRAPHICS CONVENTIONS 1-2

CHAPTER 2 INTRODUCTION TO ISAM

2.1 THE INDEXED SEQUENTIAL ACCESS METHOD 2-1
2.2 DESCRIPTION OF INDEXED SEQUENTIAL FILES 2-3
2.2.1 The Data File 2-4
2.2.2 The Index File 2-4
2.3 ISAM ACCESS MODES 2-5
2.3.1 Counted Update Mode 2-5
2.3.2 Exclusive Open Mode 2-6
2.3.3 Hints and Restrictions 2-7
2.4 CONVERTING ISAM FILES FROM AMOS TO AMOS/L FORMAT 2-8
2.4.1 Using ISMDMP and ISMBLD 2-8
2.4.2 Using a Custom Built Program 2-9
2.4.3 Converting ISAM Access Programs to Use LOKSER 2-9

CHAPTER 3 CREATING AND LOADING AN ISAM FILE WITH ISMBLD

3.1 GENERAL OPERATING INSTRUCTIONS 3-1
3.2 CREATION MODE 3-1
3.2.1 Specifying File Parameters 3-2
3.2.1.1 "Size of Key:" 3-2
3.2.1.2 "Position of key:" 3-2
3.2.1.3 "Size of data record:" 3-2
3.2.1.4 "Number of records to allocate:" 3-3
3.2.1.5 "Entries per index block:" ... 3-3
3.2.1.6 "Empty index blocks to allocate:" 3-3
3.2.1.7 "Primary Directory?" 3-3
3.2.1.7.1 "Secondary File" .. 3-3
3.2.1.8 "Data File Device?" 3-4
3.3 FILE LOADING MODE 3-4
3.3.1 Suppressing Exclusive Open Mode (the /N Switch) 3-5
3.4 CHANGING THE DATA FILE DEVICE (THE /D SWITCH) 3-5
3.5 OPTIMIZING FILE PARAMETERS 3-5
3.5.1 Entries per Index Block 3-5
3.5.2 Empty Index Blocks to Allocate 3-6

CHAPTER 4 DUMPING AN ISAM FILE WITH ISMDMP

4.1 GENERAL OPERATING INSTRUCTIONS 4-1
4.1.1 Suppressing Exclusive Open Mode 4-2
4.2 FILE DUMP MODE 4-2
4.2.1 Sample Data File Display 4-2
4.3 INDEX FILE DUMP MODE 4-2
4.3.1 Sample Index File Display 4-3

CHAPTER 5 COMPRESSING INDEX FILES WITH ISMCOM

CHAPTER 6 FILE LOCKING TECHNIQUES

6.1 ABOUT THIS CHAPTER 6-1
6.1.1 If You Want to Access ISAM through Assembly Language Programs 6-1
6.1.2 If You Want to Access ISAM through AlphaBASIC 6-2
6.2 WHAT IS FILE LOCKING? 6-2
6.3 THE AMOS AND AMOS/L FILE LOCKING TECHNIQUES .. 6-3
6.3.1 XLOCK 6-4
6.3.1.1 Exclusive Access 6-4
6.3.1.2 Shared Access 6-5
6.3.2 FLOCK 6-5
6.3.2.1 Exclusive Access 6-6
6.3.2.2 Shared Access 6-6
6.3.3 LOKSER 6-7
6.4 RECOMMENDED FILE LOCKING TECHNIQUES 6-7
6.4.1 General Hints 6-10
6.4.2 Using ISAM with a Single Key 6-11
6.4.2.1 Example 1 - Creating a Record 6-11
6.4.2.2 Example 2 - Deleting a Record 6-12
6.4.2.3 Example 3 - Updating a Record 6-13
6.4.3 Using ISAM with Multiple Keys 6-13
6.4.3.1 Example 1 - Creating a Record 6-14
6.4.3.2 Example 2 - Deleting a Record 6-14
6.4.3.3 Example 3 - Updating a Record 6-15

CHAPTER 7 WRITING AMOS ASSEMBLY LANGUAGE PROGRAMS THAT ACCESS ISAM

7.1 GENERAL CALLING SEQUENCE 7-1
7.2 INITIALIZING THE ISAM SYSTEM (.INIT) 7-2
7.2.1 User Supplied Allocation and Deallocation Routines 7-3
7.3 FINALIZING ISAM PROCESSING (.IFIN) 7-3
7.4 OPENING A FILE FOR PROCESSING (.IOPNR) 7-4

7.5 CLOSING THE FILE AFTER PROCESSING (.ICLOS) ... 7-5
7.6 LOCATING A FREE DATA RECORD (.IGTFR) 7-5
7.7 DELETING A DATA RECORD (.IDLFR) 7-6
7.8 READING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLRD) 7-7
7.9 WRITING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLWT) 7-8
7.10 FINDING A RECORD (.IREDR) 7-9
7.11 ADDING A SYMBOLIC KEY (.IWRTR) 7-10
7.12 DELETING A KEY (.IDELK) 7-10
7.13 FINDING THE NEXT SEQUENTIAL KEY (.SREDR) 7-11

CHAPTER 8 WRITING AMOS/L ASSEMBLY LANGUAGE PROGRAMS THAT ACCESS ISAM

8.1 GENERAL CALLING SEQUENCE 8-1
8.2 INITIALIZING THE ISAM SYSTEM (.INIT) 8-2
8.2.1 User Supplied Allocation and Deallocation Routines 8-3
8.3 FINALIZING ISAM PROCESSING (.IFIN) 8-3
8.4 OPENING A FILE FOR PROCESSING (.IOPNR) 8-4
8.5 CLOSING THE FILE AFTER PROCESSING (.ICLOS) ... 8-5
8.6 LOCATING A FREE DATA RECORD (.IGTFR) 8-5
8.7 DELETING A DATA RECORD (.IDLFR) 8-6
8.8 READING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLRD) 8-6
8.9 WRITING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLWT) 8-7
8.10 FINDING A RECORD (.IREDR) 8-7
8.11 ADDING A SYMBOLIC KEY (.IWRTR) 8-8
8.12 DELETING A KEY (.IDELK) 8-8
8.13 FINDING THE NEXT SEQUENTIAL KEY (.SREDR) 8-9

CHAPTER 9 STANDARD ISAM SYMBOLS FOR ASSEMBLY LANGUAGE PROGRAMMERS

9.1 CALLING SYMBOLS 9-1
9.2 CALLING FLAGS 9-2
9.3 COMPLETION CODE SYMBOLS 9-2

CHAPTER 10 USING ISAM FROM WITHIN ALPHABASIC

10.1 OPENING AN INDEXED SEQUENTIAL FILE 10-1
10.2 THE ISAM STATEMENT 10-3
10.2.1 ISAM Statement Codes 10-4
10.3 READING AND WRITING DATA IN AN ISAM DATA FILE 10-6
10.4 CLOSING FILES 10-7
10.5 LOCKING AND UNLOCKING STATEMENTS 10-7
10.6 ERROR PROCESSING 10-8
10.7 USING THE ISAM FUNCTIONS WITHIN AN ALPHABASIC PROGRAM 10-9
10.7.1 Adding Data to an Indexed Sequential File 10-9

10.7.2 Reading Data Records in Symbolic Key Order 10-11
10.7.3 Reading Data Records Randomly by Symbolic Key 10-12
10.7.4 Updating Data Records 10-12
10.7.5 Deleting a Data Record 10-13
10.8 SAMPLE ALPHABASIC ISAM PROGRAM 10-15
10.8.1 Sample ISMBLD Use on an AMOS System 10-15
10.8.2 Sample ISMBLD Use on an AMOS/L System 10-16
10.8.3 Sample AlphaBASIC Program 10-17

DOCUMENT HISTORY

INDEX

CHAPTER 1

ABOUT THIS BOOK

This manual is aimed at the experienced assembly language or AlphaBASIC programmer who wants to make use of the Alpha Micro ISAM system within his or her own programs. If you are not familiar with Alpha Micro Assembly Language, please refer to the AMOS Assembly Language Programmer's Manual, DWM-00100-43 or AMOS/L Assembly Language Programmer's Manual, DSS-10007-00. If you want information on AlphaBASIC, refer to the AlphaBASIC User's Manual, DWM-00100-01.

The purpose of this manual is threefold:

1. To give you an introduction to ISAM;

2. To discuss how you can create and access ISAM files using the various programs of the ISAM system;

3. And, to give information on how to write programs in either assembly language or AlphaBASIC to locate, update, add, and delete data in ISAM files.

To that end, we have organized this manual into the following areas:

CHAPTER 2 - INTRODUCTION TO ISAM

This chapter gives a basic introduction to the concepts behind the Indexed Sequential Access Method, ISAM, and provides some basic information on the components of the Alpha Micro ISAM system. We also discuss how to transport and convert ISAM files from an AMOS system to an AMOS/L system for programmers who are upgrading from one Alpha Micro computer system family to another.

PART I - THE ISAM UTILITY PROGRAMS

Chapters 3 through 5 discuss the ISMBLD, ISMDMP, and ISMCOM utility programs that allow you to, respectively: 1) Create and load an ISAM file, 2) Inspect and debug an ISAM file, and 3) Compress an ISAM index file.

Part II - PROGRAMMING WITH ISAM

Chapter 6 contains information on performing file locking within your programs that access ISAM files. We discuss the basic concept of file locking, and show several sample AlphaBASIC program excerpts that demonstrate the proper file locking procedures.

Chapters 7 and 8 discuss the AMOS and AMOS/L (respectively) assembly language calls your assembly language programs must use to access ISAM files. Chapter 9 provides the standard ISAM assembly language symbols used by AMOS and AMOS/L assembly language programmers.

Chapter 10 contains information on using AlphaBASIC to construct programs that access ISAM files.

1.1 COMPATIBILITY NOTES

If your system has the optional LOKSER File Locking Service System installed, note that the ISAM on your system assumes that your AlphaBASIC or assembly language ISAM access programs are using LOKSER to perform file locking; such programs must follow the ISAM file update procedures in the proper order or they will receive an "?Illegal ISAM sequence" error. Please see Chapter 6 of this manual for details.

1.2 GRAPHICS CONVENTIONS

This book conforms to other Alpha Micro publications in its use of a standard set of graphics conventions. We hope these graphics simplify our examples and make them easier for you to use.

Filespec — A file specification. Such a specification identifies a file. It usually has these elements:

Devn:Filename.Extension[p,pn]

{} — Optional elements of a command line. When these symbols appear in a sample command line, they indicate elements that you may omit from the command line, depending on the effect you want to achieve.

— Underlined characters indicate those characters that the monitor prints on your terminal display. For example, whenever you see an underlined dot, <u>.</u>, this indicates the prompt symbol that the operating system displays on your terminal when you are at monitor command level.

RET — Carriage return symbol. The RET symbols marks the place in your keyboard entry for you to press the RETURN key.

CHAPTER 2

INTRODUCTION TO ISAM

ISAM is a method for organizing and retrieving data. The name of the method (Indexed Sequential Access Method) refers to the manner in which the data is organized. The information in the ISAM data file is accessed by searching a separate index file that contains a group of symbolic keys and pointers to records in the data file with which those keys are associated. By searching several levels of indices within the index file, we can locate records in a separate file much more quickly and efficiently than if we had to search the actual data file itself. Some examples of data for which symbolic keys can be specified are:

Customer information-- the name of the customer is the key (that is, an element of the data record) on which you base your search.

Payroll-- the key is an employee number.

Inventory control-- the key is a part number.

2.1 THE INDEXED SEQUENTIAL ACCESS METHOD

Finding a convenient and efficient way to access information in a file is an important problem for a programmer. Suppose, for example, that you have a phone book of five thousand names and phone numbers. If you need to find a specific person's phone number, you can start with page number one and scan every entry in the book until you find the proper name. That process is very slow and inefficient, however, because you have to deal with so many entries. A more efficient method would involve dividing the phone book into sections, and searching only those sections that might contain the data you need instead of searching the entire data base.

Organizing your data so that it is easier and quicker to search is the main idea behind ISAM. If we were to organize the phone book in somewhat the same way as ISAM would do it, we might do this:

A. First we build a file containing one logical record for each entry in the phone book; each record consists of a person's name, an address, and a phone number. We assign each entry a number (called the relative record number or the relative key) that marks its position in the file. For example, the five hundredth entry is number 499 (the first record is number 0, not 1). (The record number is called "relative" because it marks the position of the record from the front of the file; it is not an absolute disk address.) This file that contains all of our data corresponds to the ISAM data file.

B. Next we construct a file that contains information about the data file that helps us search the data file. (The file we are constructing corresponds to the ISAM index file.) When you open a phone book, you notice that the top of each page contains two words; the first and the last names that appear on that page. These two names give you an "index" into the data on that page. So, if the two words at the top of a particular page are "PENDERGRASS-PENNINGTON," you know that the names associated with all entries on that page fall somewhere in that range. Suppose, then, that this second file we are creating contains the words at the top of the phone book pages, along with the relative record numbers of the entries that fall on each page.

   Instead of searching the entire data file, we can search this much smaller "index" file. If we want to find the entry for the name PENHALL, we can search the page indices in our index file until we find two names that PENHALL falls between. Then we can search just the data file records associated with that range of names until we find PENHALL.

   When we build an index file, we say that the file contains symbolic keys. A symbolic key is an element of a logical record on which we base our search. In this example, the symbolic key we are using is the name associated with each phone book entry. We might just as easily have set up the files so that we can base our search on phone numbers or city names.

C. We have improved our original file-search procedure, but it can be improved upon still further. We now have a data file and an index file. The index file contains one level of indices (the words at the top of each page in the phone book). The next step is to provide another level of indices within the index file.

   When you look for a name in a phone book, you first find the proper page by glancing at the names at the top of each page. Then you might look at the first and last names of each column on the page to narrow your search still further. If the name for which you are searching falls between the names at the top

and bottom of the column, you begin to search each entry in that column; otherwise, you move to the next column on the page. In the same way, our index file contains a first-level index (the names at the top of the page); then it further divides the data on the page by giving indices into subgroups of entries on the page (the first and last names in each column make up the second-level index). The final level of indices (the third-level) in the index file consists of lists of names for each column in the book along with the actual record number in our original data file that contains the entry associated with that name.

D. Journeying through the levels of indices in our index file, then, we first find the page on which the name appears, then we find the column in which the name appears, then we find the actual record number of the file in which the entire entry associated with that name appears. At no time do we ever need to search the actual data file itself.

Note that the keys in the index file are grouped alphabetically. Since we find a data record by searching the index file, the data records in the data file do not need to be arranged in any particular order. An index file may not contain duplicate keys; that is, no two data records in the data file may have the same symbolic key.

The example above discusses a data file that has one index file (called the primary index file). A data file always has one primary index file; it may also have one or more secondary index files. A secondary index file is structured in the same way as the primary index file except that it contains different symbolic keys. For example, if we want to base our search of phone book entries on phone numbers as well as names, we might construct a secondary index file that contains phone numbers.

Although we constructed the example above ourselves, the ISAM program automatically creates all data files and index files for you in response to information and file specifications that you supply.

2.2 DESCRIPTION OF INDEXED SEQUENTIAL FILES

In summary, an indexed sequential file consists of two random files: 1. the data file, containing the actual data; and 2. the index file, containing pointers to symbolic keys within the data file. You specify the location of the symbolic key within each record when you build an indexed sequential file using the ISMBLD program (discussed in Chapter 3, "Creating and Loading an ISAM File with ISMBLD"). To build an indexed sequential file, you supply certain parameters to the ISMBLD program; ISMBLD then produces an empty file.

To load the file with data, you may write your own program or you may use the ISMBLD program to copy the data from an ordinary sequential file into the data file (updating the index file in the process).

The ISAM program does all reading and writing of the index file; you will not have to handle these functions yourself. Your AlphaBASIC or assembly language program will add, delete, or update data in the ISAM data file based on the relative record number returned to your program by ISAM.

2.2.1 The Data File

The data in your data file may be in any data format; however, the index file orders keys in ASCII collating sequence (i.e., ascending binary order) which may affect operation of the ISAM program when data is recorded in other than ASCII form. When you build an indexed sequential file (via the ISMBLD program), you supply various items of information about your data file (e.g., the size of the data records, the location of the symbolic key within the data record, and so on); ISMBLD then builds both the data file and its primary index file.

Your programs use the ISAM functions to add and delete data records in the data file. When you add a record, ISAM inserts it into the first free space in your data file. When you delete a record, ISAM does so by recovering the space in the data file used by that record, and returning that area to the free record list so that it is available for new records. Because the Alpha Micro operating systems (AMOS and AMOS/L) require that contiguous files (e.g., an ISAM data file) be preallocated, once the data file is full it must be reorganized before it can be used further. For this reason, be careful to allocate as many records as you will need for the file.

All ISAM data files MUST have the extension .IDA.

2.2.2 The Index File

The ISMBLD program automatically creates the index file from a description of the data file. The index file contains three levels of indices, the lowest of which contains pointers to the records in the data file. Each successive index level points to all the blocks containing the next lower level index. Index levels are provided so that the entire index need not be searched each time a symbolic key is accessed. When a symbolic key is accessed, ISAM reads the highest level index to find which lower level index contains a pointer to the approximate location of that key. ISAM then searches the block of that lower level index; that index block in turn points to a lower index block which points to the data record in which the key is stored.

In addition to the index blocks, the index file contains another block named the Directory Rock, so called because it never moves. This block contains information describing the index and data files as well as maintenance information (e.g., free record links, access counts, etc.).

Each data file must have a primary index file; in addition to this, it may have several secondary index files. A typical example of the use of this feature would be a mailing list maintenance program, where the data is keyed on both a hashed retrieval code for unique reference and also keyed on the person's name. (For an example of this kind of program, see the sample AlphaBASIC ISAM program in Chapter 10, "Using ISAM From Within AlphaBASIC.")

Your programs use the ISAM functions to add and delete keys from the index files, and to locate data records in the corresponding data files.

The extension of an ISAM index file MUST be .IDX.

2.3 ISAM ACCESS MODES

Two access modes, Counted Update mode and Exclusive Open mode, allow ISAM to avoid unnecessary processing of your index files.

Counted Update mode is the normal access mode for assembly language or AlphaBASIC programs. (In AlphaBASIC programs, you will select Counted Update mode by opening the ISAM file in INDEXED mode; you will select Exclusive mode by opening the ISAM file in INDEXED'EXCLUSIVE mode.) Exclusive Open mode is the normal access mode for the ISMDMP and ISMBLD programs and is the only mode for the ISMCOM program. ISAM always processes indexed sequential files in one or the other of these modes. The next two sections discuss both of these modes.

NOTE: The paragraphs below mention the need for file interlocking. It is most important that your programs guard against the possibility of more than one user trying to update the same data file at the same time. If several users were to try to write to the same file record at the same time, severe damage to your data file could result. For information on file interlocking procedures, see Chapter 6 in this manual, "File Locking Techniques."

2.3.1 Counted Update Mode

Counted Update mode allows ISAM to increase its speed by avoiding any unnecessary processing. Every time ISAM updates a file in any way, it increments a counter in the Rock portion of the index file. At the time of file access, ISAM checks this counter to see if the file has been updated since the last access. If the file has not been updated, ISAM can skip further access initialization and take

advantage of its prior knowledge about the file. These actions are completely transparent to the user and the speed gains (3 to 70 times faster access times) are free.

IMPORTANT NOTE: The Counted Update mode does NOT eliminate file interlock requirements from your programs. If anyone might possibly be updating the file, your program must continue to use file interlock programs such as LOKSER, XLOCK or FLOCK to prevent simultaneous updates or accesses. See Chapter 6 for more information on file locking.

Note that it is not necessary to open and close the ISAM file with each manipulation even though the FLOCK commands are so named. It is acceptable to leave the file open during the whole interlocking and release process and is, in fact, the only way to gain the speed increase made possible by the Counted Update mode. See Section 6.4.1, "General Hints," for a discussion of factors to consider when designing your file locking procedures.

2.3.2 Exclusive Open Mode

When a program opens a file exclusively, ISAM renames the .IDX file to a .IDY extension. ISAM also sets a flag in the Rock that identifies the file as an exclusive file. If any other job tries to open that file, it receives a "?File not found" error; if another job tries to access the file once it is open, the job receives a "?Link structure smashed" error (IS.LSS).

As a result of the exclusive open, ISAM knows that no other program will be updating or accessing the file. It can therefore take full advantage of the single-process situation for initialization and change posting. Except for the process of opening the file and the need to properly close the file, use of ISAM is the same as in previous versions. The use of this mode results in an extremely large gain in access speed.

The only file interlock problem occurs at the moment of the ISAM OPEN call; no one may update the file while you are opening it. You MUST prevent this situation from occurring by using one of the file interlock programs, FLOCK or XLOCK (if LOKSER is not installed on your system), or by simply making sure that no other user is running a program that can update that file. Once your program has executed the ISAM OPEN call, your program needs no interlocks since no one else can access the file.

Invoke the Exclusive Open mode from within AlphaBASIC by using the file mode of INDEXED'EXCLUSIVE in the file OPEN statement in your program. Your assembly language program may select the Exclusive Open mode by setting the IS.EXC flag in R0 (for AMOS systems) or D0 (for AMOS/L systems) for the .IOPNR call. You MUST close the file when you are completely done with manipulating that file so that ISAM

can post the final updates and remove the Exclusive Open conditions from that file. If an error occurs during processing, you should close the file to remove the Exclusive Open conditions (although you can also remove them manually). A file in which an error occurred during updating is probably badly damaged.

Use of the Exclusive Open mode can result in significant gains when printing reports and other such batch-type operations. It does have the drawback that no one else can access the file for any reason while it is exclusively opened. Be warned that any attempts to circumvent the exclusive properties of such a file by clever manipulations will probably meet with disaster. If several people need to access the file at the same time, use the normal mode, Counted Update mode; if no one updates the file, you will lose very little speed in changing to that mode.

The ISMBLD, ISMDMP, and ISMCOM programs use the Exclusive Open mode. To prevent ISMBLD (when loading or cross-indexing an existing file) or ISMDMP from using Exclusive Open mode, use the /N switch. The /N switch must appear at the very end of the command line that invokes the program. For example:

```
.ISMBLD LABELS/N RET
```

ISMBLD (when creating/loading or creating/cross-indexing) and ISMCOM always use the Exclusive Open mode.

2.3.3 Hints and Restrictions

The two access modes make possible a dramatic increase in the speed of ISAM data accesses. They also may result in slightly peculiar situations of which you should be aware:

1. If the Counted Update mode counter has not changed, ISAM assumes that no updates have been made to the file since the last time an access was made, and that it may therefore make certain assumptions about file status and contents. The counter cycles on a count of 16,777,216. If by some very unlikely chance the file were to remain open for an incredibly long time and exactly 16,777,216 updates were made between accesses, ISAM would access and/or update the file using out-of-date information.

   Although not strictly impossible, it is very unlikely that this situation will occur. We estimate that you would have to leave your machine up and running for several weeks with the ISAM file open without making any accesses to that file in order to see this happen.

2. When you open a file in Exclusive Open mode, ISAM must be able to write to the disk that contains the file. This

means that you must make sure that the disk is not write-protected even if all accesses to that file are going to be read operations.

3. The most visible quirk of the Exclusive Open mode is that it causes ISAM to rename the extension of the file being opened from .IDX to .IDY. If such a file is not properly closed (for whatever reason), then the name of that file will not be correct in the disk directory. You can cure this problem very easily by using the RENAME command. For example:

```
.RENAME *.IDX=*.IDY(RET)
```

A file OPEN also changes a flag in the Rock of the ISAM index file. You do not need to worry about changing the flag yourself in the event of an improperly closed file, since the situation is automatically self-correcting the next time the file is opened (in either Exclusive Open or Counted Update mode).

4. If you open a file in Counted Update mode but the file was last used in Exclusive Open mode and was never closed, the file OPEN will cause ISAM to write to the file to correct the exclusive flag. If this situation is going to occur, make sure that the disk is write-enabled.

2.4 CONVERTING ISAM FILES FROM AMOS TO AMOS/L FORMAT

This section contains information on transferring ISAM files from AMOS systems to AMOS/L systems, and vice versa. Because the format of ISAM files is different on AMOS systems (AM-100 or AM-100/T CPU based systems) than on AMOS/L systems (AM-100/L CPU and AM-1000 based systems), moving ISAM files between these systems requires a translation process. Note that you only need to use these procedures when transferring ISAM files between systems that are based on different processors; you may move ISAM files between systems based on the same processor without using any special procedures.

2.4.1 Using ISMDMP and ISMBLD

The simplest method of transferring ISAM files involves the use of the ISMDMP and ISMBLD programs. However, this method imposes some restrictions on the way data within the ISAM file is structured. The records in the data file may contain only ASCII data, and the key must be contained within the data record. These restrictions actually follow recommended guidelines since they greatly simplify rebuilding and recovery of ISAM files in case of catastrophe.

To transfer an ISAM file using this method, first use ISMDMP on the system you are moving data from to dump the contents of the ISAM file to a .SEQ file. (See Chapter 4 for further information on ISMDMP.) Once these sequential data files have been created, move them to the new system using any of the standard transfer media (removable disk pack, floppy disk, VCR cassette, etc.). Once the sequential files are on the new system, you must create new ISAM files using the ISMBLD program on the new system. Use ISMBLD to load the sequential files into the ISAM files. You can load each of the secondary index files in the same fashion.

2.4.2 Using a Custom Built Program

Because the ISAM system is so flexible, it allows the creation of files that cannot be transferred by the standard method outlined above. The most common type of file in this category will be the file whose symbolic keys are not stored within the data record. (One of the many reasons why constructing files of this type is poor practice is that it makes the transferring of ISAM files more cumbersome.)

To transfer these non-standard ISAM files, you must create a custom AlphaBASIC program that outputs the contents of the ISAM file to a random file that can then be transferred to the new system. Another custom program on the new system must then load this random file into the new ISAM file.

To get the key associated with a data record, if that symbolic key is not contained within the data record, you must use the ISAM 7 call within AlphaBASIC.

2.4.3 Converting ISAM Access Programs to Use LOKSER

See Chapter 6 for information on the File Locking Service System, LOKSER, available with AMOS 5.0 and AMOS/L 1.1 releases and later. If you want to convert your programs from performing file locking using XLOCK and FLOCK to the format used by LOKSER, refer to the AMOS LOKSER User's Manual, DSS-10022-00, or the AMOS/L LOKSER User's Manual, DSS-10034-00.

PART I

THE ISAM UTILITY PROGRAMS

The next few chapters discuss the ISAM utility programs. These programs: 1. create and (optionally) load an indexed sequential file; 2. display the contents of your data and index files; and, 3. allow more efficient use of your index files by compressing index block entries.

CHAPTER 3

CREATING AND LOADING AN ISAM FILE WITH ISMBLD

The ISMBLD program provides a convenient method for creating and loading indexed sequential files. It gives you the ability to create a new indexed sequential file, to add records to the data file from an ordinary sequential data file, and to create a secondary index file that cross-indexes to a primary index file.

3.1 GENERAL OPERATING INSTRUCTIONS

ISMBLD has three operating modes: 1. create a new indexed file; 2. add data to the new file or to an existing file; and, 3. change the device specification of a data file. All modes are called via the general command:

```
.ISMBLD filespec{/D}{/N} RET
```

If the indexed sequential file specified by filespec does not exist, ISMBLD enters the creation mode. If the file already exists, ISMBLD enters the data loading mode unless you have specified the optional /D maintenance switch. (NOTE: If the file already exists, you may specify Counted Update mode by using the /N switch. See Section 3.3.1, "Suppressing Exclusive Open Mode.")

3.2 CREATION MODE

The creation mode is the most commonly used mode. In this mode you input series of parameters that describe the desired indexed sequential file. From these parameters the ISMBLD program generates a data file/primary index file combination or a secondary index file that cross-indexes to an existing primary index file.

3.2.1 Specifying File Parameters

Before actually creating the file, ISMBLD asks you a number of questions about your data file. In response to each of the questions, you are expected to enter a valid answer. Because of the myriad ways that you can set up an indexed sequential file, very little validity checking is done on your answers. It is therefore possible to create totally useless files. Be careful. For an example of the ISMBLD dialog, turn to Chapter 10, "Using ISAM From Within AlphaBASIC."

The following sections describe the questions asked and the expected responses:

3.2.1.1 "Size of Key:"

Enter the size of the desired key in decimal bytes. To minimize index search time, keep this size as small as possible. The maximum key size is 256. When you later access the ISAM files you are now creating, you must remember to pad with blanks or other characters keys that are smaller than this specified size. Pad numeric fields in the front of the field; pad symbolic keys at the end. One side effect of this is that both binary and floating point keys may be used.

3.2.1.2 "Position of key:"

This parameter specifies the location of the key within the data record. The symbolic key position is used when loading indexed sequential files from sequential files as the means of determining the symbolic key. Enter the number of the first character-position in the record which the key occupies; the first position within a record is position number one.

3.2.1.3 "Size of data record:"

This parameter defines the size of the records in the data file or the maximum data record size in the case of variable length records. Specify this size in bytes (decimal). The data record size must be greater than or equal to the key size plus the key position.

3.2.1.4 "Number of records to allocate:"

This parameter defines the number of records that the data file is to contain. (NOTE: On AMOS/L systems, after you answer this question, ISMBLD tells you how many index blocks it is automatically allocating for you. If this number is not sufficient, you may specify additional blocks in response to the question "Empty index blocks to allocate:"-- see Section 3.2.1.6.)

3.2.1.5 "Entries per index block:"

This parameter allows you to specify the number of entries contained in an index block; this value can greatly affect the efficiency of searches and inserts within the file. See Section 3.5, "Optimizing File Parameters," for more information.

NOTE: On AMOS/L systems, ISMBLD does not ask this question, since it automatically calculates the entries per index block for you.

3.2.1.6 "Empty index blocks to allocate:"

ISMBLD allocates for you the bare minimum number of index blocks you will need to contain keys for the specified number of data records. This calculation is based on the assumption that the index file tree structure will be perfectly balanced. Since this is rarely the case, you will probably need to specify an additional number of index blocks.

3.2.1.7 "Primary Directory?"

If you are creating a primary index and data file combination, enter Y; if you are creating a secondary index file, enter N.

3.2.1.7.1 "Secondary File"

If you are building a secondary index file, ISMBLD prompts you for the file specification of the primary index file:

Secondary index to file:

Enter the specification of the primary index file to which this secondary file cross-indexes. Type just a RETURN to exit ISMBLD. You may create as many secondary index files as you want that cross-index to a particular primary index file by re-invoking ISMBLD with the specification of that primary index file and specifying a new secondary index file.

If you have created a secondary index file, your dialog with ISMBLD is now over. ISMBLD returns you to monitor command level. If you are creating a data file/primary index file combination, ISMBLD asks you for more information (see below).

3.2.1.8 "Data File Device?"

ISMBLD now asks you:

Data File Device?

If the data file is to be on a different device than the index file, enter the name (and number) of that device. If they are to be on the same device, enter a RETURN. For example, if the data file is to be on unit 1 of device "DSK," enter:

Data File Device? DSK1: RET

3.3 FILE LOADING MODE

After an indexed sequential file has been created, it is often desirable to load the data and index files with data from an ordinary sequential data file. To allow this, ISMBLD enters the data loading mode once it creates the indexed sequential file.

If you want to load data into an existing data file, invoke ISMBLD with the name of that file. ISMBLD then responds:

[Processing existing file]

This notifies you that you are in the file loading mode and not the creation mode.

ISMBLD now prompts you for a sequential file specification by typing:

Load from file:

You may now enter the file specification that selects the sequential data file from which you want to load. A default extension of .SEQ is assumed by ISMBLD. (If you do not want ISMBLD to load the new file for you or if you have made an error in the file specification you gave to ISMBLD, type a RETURN after the "Load from file:" prompt; no data will be added to the data file.)

3.3.1 Suppressing Exclusive Open Mode (the /N Switch)

When loading an existing file, ISMBLD normally uses Exclusive Open mode. If you wish it to use Counted Update mode instead, include the /N switch at the end of the ISMBLD command line. For example:

```
.ISMBLD MAIL/N (RET)
```

3.4 CHANGING THE DATA FILE DEVICE (THE /D SWITCH)

The only creation data that you can change is the data file device. The /D switch provides this field for examination and change. Simply enter the new device name or a RETURN (to leave the device unchanged). To change the device to the same device that the index file uses, enter a period (.) only. It is your reponsiblity to move the file to the specified device.

3.5 OPTIMIZING FILE PARAMETERS

This section provides some hints on how to organize an indexed sequential file for maximum efficiency.

Once your file has stabilized and you aren't changing it much, re-evaluate the original file parameters. If your evaluation so indicates, rebuild the file with different parameters.

3.5.1 Entries per Index Block

NOTE TO AMOS/L USERS: ISMBLD automatically calculates this value for you. This section does not apply to your system-- skip down to Section 3.5.2, "Empty Index Blocks to Allocate."

This parameter is a two-edged sword. A small value means faster in-core searches, but more disk accesses and more block splits during record additions. A large value reduces the number of disk accesses and block splits, but increases in-core search time and increases the amount of memory used for buffers. (A block split occurs if you add a key to an index block, but there is no more room in that block; ISAM automatically "splits" that block and redistributes the keys among the two new blocks.)

Since the index structure is fixed at three levels deep, the maximum number of keys that you may add to an index without the top index block splitting is n^3, where n is the number of entries per index block. When the top index block splits, the search time through the index increases due to the possibility of having to do more disk reads.

When you use a floppy disk, the in-core search time is so small compared to a disk seek/transfer that any increase/decrease will not be apparent. When you use a faster disk the trade-off becomes trickier. As a rule, keep the number of entries as large as possible, consistent with the user memory partition size. The amount of index buffer space required is:

5 * ((entries-per-block * (keysize + 4))+2)

where key size is rounded to an even number of bytes. Given this, you should be able to determine a reasonable value for the number of entries. (NOTE: (keysize + 4) * entries-per-block MUST be less than or equal to 510.) The more entries per block, the more memory you use. It is sometimes more efficient to have the top block split a few times rather than to eat up a large amount of memory.

3.5.2 Empty Index Blocks to Allocate

During creation, enough index blocks are allocated to support a balanced index file tree with sufficient nodes for the number of data records allocated. In practice, the index file tree is rarely balanced (unless you add records in a truly random number with an even distribution of key values). Because of this, you should allocate empty index blocks.

For AMOS systems, a good number of empty index blocks to allocate is the total number of data records divided by the number of entries in an index block.

For AMOS/L systems, doubling the number of index blocks allocated automatically by ISAM is a good number to use. To do this, simply respond to the "Empty index blocks to allocate" question with the number ISMBLD displayed after you entered the number of records to allocate.

CHAPTER 4

DUMPING AN ISAM FILE WITH ISMDMP

The ISMDMP program provides a convenient method for unloading an indexed sequential file into a sequential file. It also provides a means of examining the index file structure to determine how balanced that structure is.

NOTE: ISMDMP works the same on AMOS and AMOS/L systems.

4.1 GENERAL OPERATING INSTRUCTIONS

ISMDMP has two operating modes: the first allows you to output the contents of an indexed sequential file to an ordinary sequential file; the second allows you to display the index file structure on a terminal to allow analysis thereof. Both are invoked via the general command form:

```
.ISMDMP filespec{/N}(RET)
```

where filespec specifies an indexed sequential file and the optional /N switch suppresses Exclusive Open mode. (See below.) After performing some initialization procedures, ISMDMP asks:

```
Output to:
```

Supply another file specification; this one selects the sequential output file. ISMDMP assumes a default file extension of .SEQ. If you want to enter the index file dump mode, enter TTY: as the file specification. For example:

```
Output to: TTY:(RET)
```

4.1.1 Suppressing Exclusive Open Mode

ISMDMP normally uses Exclusive Open mode when performing its file accesses. If you wish it to use Counted Update mode instead, use the /N switch at the end of the ISMDMP command line. For example:

```
.ISMDMP STAT/N(RET)
```

4.2 FILE DUMP MODE

In this mode, ISMDMP outputs the records of the indexed sequential file to an ordinary sequential file in ascending key order. ISMDMP does no translation of the records; it outputs the records in exactly the same form as they were input at some earlier date.

4.2.1 Sample Data File Display

We used ISMBLD to create a small ISAM data file named LABELS. Then we used the sample program in Chapter 10, "Using ISAM From Within AlphaBASIC," to place five records in the file. We then asked ISMDMP to place the data in that file into a file named DATDMP:

```
.ISMDMP LABELS(RET)

Output to: DATDMP(RET)

5 records dumped
.
```

If we use the TYPE command to display the new file (e.g., TYPE DATDMP.SEQ), we see:

FILMORE SUSAN	230 STILWOOD LOWELLMA15673200
HINCHEY EDSEL	6712 VIA MALAGA TUSTINCA90245102
LAWRENCE T.E.	1023 W. SANDS PANGUITCHUT98344100
MUKLUK, H.	345 PRAIRIE DOG LN BAKERCA98766120
SAVOY JOHN	891 E. DECATUR LAS VEGASNE89023103

Each record contains: 1. Customer name; 2. street address; 3. city; 4. state (two letters); 5. zip code; and, 6. three-digit identifying number (called a hash number).

4.3 INDEX FILE DUMP MODE

The dump mode is intended primarily as a debugging tool, and will not find much use among general users. Therefore we provide little documentation on its use. Those of you who understand the basic structure of the index file should be able to figure out the display quite easily.

```
                                          000000000000
        000000                  1777777
000003: FILMORE SUSAN                     000000000414 [points to data record]
        HINCHEY EDSEL                     000000177777 [first record entered]
        LAWRENCE T.E.                     000000000206
        MUKLUK H.                         000000000311
        SAVOY JOHN                        000000000103
        --------------------------        177777177776 [indicates last record]
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
        000000                  044506    [in used blocks, this number is
                                          junk-- ignore it.]
000004:                                   000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
        000000                  000005    [in unused blocks, points to next
                                          free index block]
000005:                                   000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
                                          000000000000
        000000                  000006
                  .
                  .
                  .
                  .
                [Etc.]
```

CHAPTER 5

COMPRESSING INDEX FILES WITH ISMCOM

NOTE: This command is not used on AMOS/L systems.

ISMCOM.PRG compresses the upper level of ISAM index files: this increases access speed and may recover some storage room in the index file. To use ISMCOM, enter:

```
.ISMCOM Filespec (RET)
```

where filespec selects the index file you want to compress. The program now reports its intended compression factor (initially based on 95%). If you wish denser or looser compression, enter the percentage of compression you want ISMCOM to use. If that value is valid for the file (based on the number of entries per index block), the program proceeds: otherwise, it reports the actual effective value and allows you to enter a new value. The only way to get 100% compression is to enter 100. The program will not accept input of a percentage of less than 50%. (In actual practice, 50% can be rounded down to, say, 47% in some cases.) Below is a sample ISMCOM dialog:

```
.ISMCOM DATA.IDX (RET)

NOBODY else may use this file while I'm processing it.

I am planning to compress each block to at least 90 percent full
If that is not acceptable, enter the percentage you desire: 76 (RET)
It will actually work out to be 80% full
If that is not acceptable, enter the percentage you desire: (RET)

No blocks unchanged.  No blocks freed.  No blocks compressed.
```

Note that a compression factor of 100% will cause a block split the next time a top level index is created. The number 95% was chosen as the optimum compression factor for most files. At the end of compression, ISMCOM prints some statistics that tell you how much compression was done and how much good it should do.

PART II

PROGRAMMING WITH ISAM

This section contains information on writing assembly language programs and AlphaBASIC programs that use the ISAM functions to access and update ISAM files.

The first chapter discusses general file locking techniques, with emphasis on AlphaBASIC programming.

For information on writing assembly language programs on the AMOS and AMOS/L systems, refer to:

AMOS Assembly Language Programmer's Manual, DWM-00100-43
AMOS Monitor Calls, DWM-00100-42
AMOS/L Assembly Language Programmer's Manual, DSS-10007-00
AMOS/L Monitor Calls, DSS-10003-00

For information on AlphaBASIC, refer to:

AlphaBASIC User's Manual, DWM-00100-01

For information on the LOKSER File Locking Service System on AMOS and AMOS/L systems, and for instructions on converting AlphaBASIC programs to use the LOKSER method of file locking, refer to:

AMOS LOKSER User's Manual, DSS-10022-00
AMOS/L LOKSER User's Manual, DSS-10034-00

CHAPTER 6

FILE LOCKING TECHNIQUES

6.1 ABOUT THIS CHAPTER

This chapter contains information on the following topics:

* What is file locking and why is it important?
* What are the file locking alternatives?
* What are the recommended procedures for locking ISAM files and updating data?

Because most programmers will be using LOKSER from within AlphaBASIC programs to control ISAM file access, this chapter uses sample AlphaBASIC programs that use LOKSER to illustrate the recommended procedures for locking ISAM files and performing data updates.

6.1.1 If You Want to Access ISAM through Assembly Language Programs

If you are not experienced with using ISAM from within assembly language programs, you may want to read Chapters 7 and 8 of this manual to familiarize yourself with the assembly language/ISAM programming interface for AMOS and AMOS/L systems before continuing on with this chapter.

If you access ISAM through assembly language programs, the techniques discussed in this chapter apply to you, even though all examples are illustrated with AlphaBASIC programs-- Use the AMOS or AMOS/L monitor calls that are equivalent to the AlphaBASIC statements discussed below to perform the same procedures. For example, the AMOS INPUTL monitor call performs the same function as the AlphaBASIC READL statement-- it inputs and locks a record that will later be updated.

Chapters 7 and 8 discuss the assembly language calls that are equivalent to the AlphaBASIC ISAM statements for locating, creating, and deleting records in an ISAM file. For information on the file

locking monitor calls, see the section "File/Record Locking Service (LOKSER) Monitor Calls," in Chapter 6, "The File Service System," in the AMOS Monitor Calls, DWM-00100-42, or the AMOS/L Monitor Calls, DSS-10003-00.

If your system does not have LOKSER installed, you will want to write your own file locking routines to prevent simultaneous update of ISAM files. The sections below, however, will give you guidelines on the proper order in which to perform file locking and data update steps.

6.1.2 If You Want to Access ISAM through AlphaBASIC

If you are not already experienced in using ISAM from within AlphaBASIC programs, you may want to read Chapter 10 of this manual to familiarize yourself with the AlphaBASIC/ISAM programming interface before continuing with this chapter.

If your system does not have LOKSER installed, then you will want to use XLOCK or FLOCK (discussed in Section 6.3) to perform file locking. You will want to read the examples in section 6.4, however, for information on the correct order in which to perform file locking and data updates steps. For information on converting AlphaBASIC programs that use FLOCK or XLOCK to use LOKSER, refer to the AMOS LOKSER User's Manual, DSS-10022-00, or the AMOS/L LOKSER User's Manual, DSS-10034-00.

6.2 WHAT IS FILE LOCKING?

One potential problem for multi-user systems is the possibility that two users will access the same file at the same time and try to simultaneously update information in that file, causing data loss. For example:

1. I read a data record;

2. You read the same data record;

3. I change the data in one field of that record, and write the record out;

4. You change data in the same record and write the record out;

My change disappears when you write out your copy of the record because when you originally read the record, my change had not yet been made.

Of course, we are not aware that we are both using the same data record-- keeping track of exactly what records to update is an internal procedure performed by the programs we are using.

In order to prevent this kind of simultaneous update problem, it is necessary to perform "file locking." That is, when I need to update a record, the program I am using locks the entire file or just the record so that nobody else can access it until I am done. When another program tries to access the file while it is locked, the program is told that the file is already in use.

This chapter discusses the various methods available to your programs on AMOS and AMOS/L systems for performing file locking.

6.3 THE AMOS AND AMOS/L FILE LOCKING TECHNIQUES

For AMOS Versions 4.6A and AMOS/L Versions 1.0A and earlier, we provide two techniques for file locking from within AlphaBASIC programs: using the AlphaBASIC XLOCK or FLOCK assembly language subroutines.

As of AMOS Versions 5.0 and AMOS/L Versions 1.1 and later, a third alternative is available: LOKSER, the File Locking Service System, which provides automatic file locking not only from within AlphaBASIC programs, but at a system level as well. To see why this is important, consider the fact that FLOCK and XLOCK are not true security devices, but rely upon all programs accessing data files to "follow the rules" in checking to see if a file is in use. A user program can easily circumvent the file locking by simply not using FLOCK or XLOCK to see if someone else is using the file. In addition, FLOCK and XLOCK cannot check for deadlock (a situation in which two programs come to a standstill because each needs to access a file the other has already locked.)

LOKSER, on the other hand, provides a system level file locking system. AlphaBASIC, AlphaVUE, or any other program running on an AMOS or AMOS/L system goes through the operating system when it wants to access a file. On a system on which LOKSER is installed, the operating system always checks with LOKSER before accessing a file to see if the file is available. If LOKSER knows that a file is in use, it will not allow another program or user to access it. If the System Operator requests it, LOKSER will even check to make sure that all programs and users access specific files in a specific order to eliminate the possibility of deadlock.

Note that even if your version of AMOS or AMOS/L supports LOKSER, installing LOKSER is optional. If you wish, you may continue to use XLOCK or FLOCK instead of installing LOKSER. However, LOKSER provides a much simpler and more error resistant technique for file locking. For complete information on using XLOCK, FLOCK, and LOKSER, refer to the AMOS LOKSER User's Manual, DSS-10022-00, or the AMOS/L LOKSER User's Manual, DSS-10034-00. Those manuals also contain a comparison of the three methods and examples of their use.

6.3.1 XLOCK

XLOCK is an assembly language subroutine callable from an AlphaBASIC program. Although it has the disadvantage that its use is voluntary (that is, it controls file access only if all parties involved use it to see whether or not a file is in use), XLOCK does offer the advantages of low overhead and high speed. XLOCK has a major weakness-- it locks on a file channel rather than on an actual file specification. That means that other programs using XLOCK to access a locked file can indeed access the file if they open it on a file channel other than the one it was originally locked on. Or, if a program accidently opens another file on a file channel that was previously locked, the new file becomes locked.

XLOCK has another limitation-- XLOCK does not technically allow you to lock an entire file for Exclusive access, although you can simulate such a lock by locking all records in the file.

6.3.1.1 Exclusive Access

Below is an example of using XLOCK within an AlphaBASIC program to lock a file for Exclusive access. Because all records in the file are locked, no individual record locks are necessary.

```
10      MAP1  XLOCK'PARS            ! XLOCK Parameters
20          MAP2  XLOCK'MODE,B,2    !   Mode/status return
30          MAP2  XLOCK'FILE,B,2    !   File channel number
40          MAP2  XLOCK'BLOCK,B,2   !   Block number

        { Set up code }

100     XLOCK'MODE = 0           ! Lock or return error
110     XLOCK'FILE = FILE'NO     ! File channel number
120     XLOCK'BLOCK = 0          ! Wild card - lock entire file
130                              !   by locking all records
140                              ! Lock the file for Exclusive use.
150     XCALL XLOCK, XLOCK'MODE, XLOCK'FILE, XLOCK'BLOCK
160     IF XLOCK'MODE <> 0 &
           THEN GOTO XLOCK'ERROR   ! Other user already has it?
170                                ! Nobody's got it-- OK to open.
180     OPEN #FILE'NO, FILE'NAME, INDEXED'EXCLUSIVE, &
                REC'SIZE, REC'NUM

        { Get, process, and output record }
```

```
300     CLOSE #FILE'NO          ! Close and update last record
310     XLOCK'MODE = 2          ! Clear the lock
320                             ! Release the file.
330     XCALL XLOCK, XLOCK'MODE, XLOCK'FILE, XLOCK'BLOCK

        { Finish up }
500 XLOCK'ERROR:
        { Error processing }
```

6.3.1.2 Shared Access

Below is an example of using XLOCK from within an AlphaBASIC program to lock a file for Shared use. In this case you lock the entire index file via XLOCK, then READ the record, modify the data, WRITE the record, and unlock the index file.

```
10      MAP1  XLOCK'PARS                ! XLOCK Parameters
20         MAP2  XLOCK'MODE,B,2         !   Mode/status return
30         MAP2  XLOCK'FILE,B,2         !   File channel number
40         MAP2  XLOCK'BLOCK,B,2        !   Block number
50      XLOCK'FILE = FILE'NO            ! File channel number
60                                      ! Open the file.
70      OPEN #FILE'NO, FILE'NAME, INDEXED, REC'SIZE, REC'NUM
80 RECORD'LOOP:
90      XLOCK'BLOCK = 0                 ! Wildcard to protect index
100     XLOCK'MODE = 0                  ! Mode = Lock or return
110                                     !   if error.
115                                     ! Lock the index.
120     XCALL XLOCK, XLOCK'MODE, XLOCK'FILE, XLOCK'BLOCK
130     IF XLOCK'MODE <> 0 &
           THEN GOTO XLOCK'ERROR        ! Is index available?

        { Get, process, and output record }

200     XLOCK'MODE = 2                  ! Mode = Clear lock
210                                     ! Clear the index lock
220     XCALL XLOCK, XLOCK'MODE, XLOCK'FILE, XLOCK'BLOCK
230     GOTO RECORD'LOOP                ! Get next record.
240 XLOCK'ERROR:
        { Error processing }
```

6.3.2 FLOCK

FLOCK is an improved version of XLOCK. It allows you to lock a file for Exclusive and Shared access, and also allows record locks. As with XLOCK, its use by AlphaBASIC programs is voluntary, and it locks on a file channel rather than on a file specification.

6.3.2.1 Exclusive Access

No record locks are required, since the entire file is locked.

```
10      MAP1 FLOCK'RETURN,F              ! FLOCK return code

        { Set up code }

100                                      ! Lock the file for
110                                      ! Exclusive access
120     XCALL FLOCK, 0, 6, FLOCK'RETURN, FILE'NO, 0
130     IF FLOCK/RETURN <> 0 &
           THEN GOTO FLOCK'ERROR         ! Already in use?
140                                      ! Nobody's got it--
150                                      ! OK to open.
160     OPEN #FILE'NO, FILE'NAME, INDEXED'EXCLUSIVE, REC'SIZE, REC'NUM

        { Process records }

200     CLOSE #FILE'NO                   ! Close and update last record
210                                      ! Release the file
220     XCALL FLOCK, 1, 0, FLOCK'RETURN, FILE'NO, 0

        { Finish up }
400 FLOCK'ERROR:
        { Error processing }
```

6.3.2.2 Shared Access

```
10      MAP1 FLOCK'RETURN,F              ! FLOCK return code
20                                       ! Request shared file
30      XCALL FLOCK, 0, 0, FLOCK'RETURN, FILE'NO, 0
40      IF FLOCK/RETURN <> 0 &
           THEN GOTO FLOCK'ERROR         ! Already in use?
50                                       ! Nobody's got it--
60                                       ! OK to open.
70      OPEN #FILE'NO, FILE'NAME, INDEXED, REC'SIZE, REC'NUM
80  RECORD'LOOP:
90                                       ! Lock the record
100     XCALL FLOCK, 3, 6, FLOCK'RETURN, FILE'NO, REC'NUM
110     IF FLOCK'MODE <> 0 &
           THEN GOTO FLOCK'ERROR         ! Is record in use?
120     READ #FILE'NO, DATA'MAP          ! Record is free-- read it

        { Process data in record }

200                                      ! Lock all records
210                                      !   in the file
220     XCALL FLOCK, 4, 6, FLOCK'RETURN, FILE'NO, REC'NUM

        { Update index entries }
```

```
400                                     ! Release all the records
410     XCALL FLOCK, 6, 0, FLOCK'RETURN, FILE'NO, REC'NUM
420     WRITE #FILE'NO, DATA'MAP        ! Output the new data record
430                                     ! Clear the data record lock
440     XCALL FLOCK, 5, 0, FLOCK'RETURN, FILE'NO, REC'NUM
450     GOTO RECORD'LOOP                ! Get next record
460  ALL'DONE:
        { Finish processing }

600     CLOSE #FILE'NO                  ! Close the file
610                                     ! Unlock the file
620     XCALL FLOCK, 1, 0, FLOCK'RETURN, FILE'NO, 0

        { Finish up }
800  FLOCK'ERROR:
        { Error processing }
```

6.3.3 LOKSER

As of AMOS Version 5.0 and AMOS/L Version 1.1 and later, another technique for file locking is available: LOKSER, the File Locking Service System. The optional LOKSER system provides automatic file locking for all files on the system or for only those files designated by the System Operator.

LOKSER also allows you to customize the file access rules for your system. For example, the System Operator can configure LOKSER to: prevent users from renaming or deleting certain files, force programs and users to open certain files in a certain order to prevent deadlock, prevent users and programs from reading files outside of the user's project accounts except for the system accounts, perform automatic file locking on all files on the system or for only certain files, etc. Please see the AMOS LOKSER User's Manual, DSS-10022-00, or the AMOS/L LOKSER User's Manual, DSS-10034-00, for details on LOKSER's features and operation.

6.4 RECOMMENDED FILE LOCKING TECHNIQUES

Because most users will be using LOKSER to control ISAM file access from within AlphaBASIC, this section uses sample AlphaBASIC programs that interface with LOKSER to illustrate the recommended procedures for locking ISAM files. However the procedures outlined below also apply to assembly language programs-- use the file locking monitor calls that are equivalent to the AlphaBASIC statements discussed below.

IMPORTANT NOTE: Under AMOS Versions 5.0 and AMOS/L Versions 1.1 and later, when LOKSER is installed, the level of error checking performed by the monitor, AlphaBASIC, ISAM, and LOKSER, is significantly more thorough than in previous system software releases. Because of this, some AlphaBASIC programs that appeared to execute correctly under

previous releases will produce errors under the later software releases. In order to understand the problems that you may face, the following discussion describes the locking/unlocking performed by the AlphaBASIC statements used to access ISAM files.

If your system does not have LOKSER installed, all of your existing AlphaBASIC programs written under previous software releases will execute under AMOS 5.0 and AMOS/L 1.1 without any changes. Without LOKSER, no checking is done for simultaneous use of a file by multiple users or correct use of the file by a single user. However, even if LOKSER is not installed, we recommend that you follow the procedures discussed below (using READL and WRITEL statements where appropriate)--even though automatic file locking will not be performed for you, your programs will be in the proper form so that later conversion to a LOKSER system will be easy.

If LOKSER is installed, you must follow the procedures outlined below; i.e., each step must be executed in the proper sequence or else you will get an "?Illegal ISAM sequence" error. You must follow this sequence whether or not you are using INDEXED or INDEXED'EXCLUSIVE modes.

The basic concept that LOKSER and ISAM use to prevent multiple users from interfering with each other during ISAM file accesses is to automatically lock the primary index (.IDX) file for Exclusive use at the start of a critical procedure and to unlock it at the end of the procedure. Intermediate steps in the procedure check to make sure that the index file is locked. The following table shows the locking/unlocking operations performed by the individual steps:

AlphaBASIC STATEMENT	OPERATION	PRIMARY INDEX FILE	SECONDARY INDEX FILE
ISAM 1,2,7	Locate record	Locked	Locked at start; unlocked at finish
ISAM 3	Add key	Must be locked Unlocked at finish	Locked at start; unlocked at finish
ISAM 4	Delete key	Must be locked	Locked at start; unlocked at finish
ISAM 5	Get data record	Must be locked	Illegal operation
ISAM 6	Release data record	Must be locked; unlocked at finish	Illegal operation
READ	Input record	Must be locked; unlocked at finish	Illegal operation
READL	Input record for update (Data record remains locked)	Must be locked	Illegal operation
WRITE	Output modified record (Unlocks data record)	Must be locked; unlocked at finish	Illegal operation
WRITEL	Output new record (Unlocks data record)	Must be locked	Illegal operation
LOCK	Lock primary index	Locked	Illegal operation
UNLOKR	Unlock record	Unlocked	Illegal operation

(When we say "Locked" or "Unlocked" in the table above, we are describing the actions ISAM automatically performs on the index file. When we say "Must be locked," we mean that the indicated statement may not explicitly perform this action, but that this condition must exist in order for successful completion of the operation. Usually, if you follow the procedures discussed below, these necessary conditions will already have been set up by previous operations. But, sometimes you must explicitly lock the primary index file yourself. See Section 6.4.1, "General Hints," below, for more information.)

The locking and unlocking are designed to occur at the proper places to allow the procedures shown below (create a record, delete a record, update a record) to execute correctly. Whether or not LOKSER is

installed on your system, it is good programming practice for your programs to perform the procedures discussed below in the order shown--If LOSKER is installed on your system, your programs MUST use the recommended procedures in the order shown, or your programs will not execute correctly.

6.4.1 General Hints

There are a couple of situations that arise because of the automatic locking procedures that you must understand in order to correctly use the automatic locking and unlocking.

Since ISAM allows you to have more than one record in a physical disk block, and LOKSER locks physical disk blocks rather than logical records, locking a record will lock all logical records in the same physical disk block, preventing other users from accessing logical records within the block you have locked. So, try not to unnecessarily lock a record any longer than you have to.

In order to understand how LOKSER and ISAM work together, you must understand what resources are actually locked. When you initiate a sequence of steps to access an ISAM file, ISAM requests LOKSER to lock the index (.IDX) file for exclusive use and wait if it is not available (i.e., wait if it is locked by another user). By using exclusive locks, ISAM obviates the need to keep record locks on the index file, thereby reducing to a minimum the overhead to access ISAM files. As the table above shows, those statements which begin an ISAM access sequence lock the index file; those statements that occur in the middle of an ISAM access sequence check to make sure the index is locked and those statements that terminate an ISAM sequence unlock the index file. In all cases, when you access a secondary index file, it is locked and unlocked by each statement.

In general, when an ISAM access statement fails because of an error, ISAM will unlock the index file before it returns to the user with an error code. There are several exceptions to this rule, however. First, when you issue an ISAM 1 call to seek a record, ISAM will leave the primary index file locked if it detects an error 33, "?Record not found." This is so you can add the record without letting another user potentially add it between the time you seek it and the time you actually attempt to add the key (ISAM statement 3). If you are not going to add the record, you must unlock the primary index via an UNLOKR statement. The second exception occurs when an error is detected while using a secondary key. ISAM cannot access the primary index and thus you must always explicitly UNLOKR the primary index file in this case.

You will notice in all of the examples below that error processing is an important part of your ISAM access program. Two main types of errors must be detected and handled: "hard" errors detected by AlphaBASIC (e.g., a "?record in use" error), and "soft" errors

detected by ISAM (e.g., "?Index is full"). The main difference between the AlphaBASIC hard errors and the ISAM soft errors is that if your program does not intercept and handle a hard error, AlphaBASIC interrupts your program and returns its user to monitor command level. If your program does not intercept and handle a soft error, AlphaBASIC does not report the error or interrupt the program, but severe damage can result to your ISAM files. You can see that it is very important for your programs to look for and handle both hard and soft errors.

Note that you use ERF(#file-channel) function to detect ISAM errors. If you want to PRINT out the value of the ERF function, do so before performing an UNLOKR statement; UNLOKR causes the ERF function to return a zero. You use the ERR(x) function to detect hard errors.

For a list of all ISAM soft error codes, see Section 10.6 of this manual. For a list of all hard (AlphaBASIC) error codes, and for information on the ERR(x) function and the ON ERROR GOTO and RESUME hard error trapping statements, see Chapter 17 of the AlphaBASIC User's Manual. You will notice that the ERF and ERR functions can return the same error codes-- these do represent different errors, however! For example, an ERR error #36 is "?Invalid Subroutine version," while an ERF error #36 is "?Index file is full." Your interpretation of the error code depends on whether the error code was returned by ERF or ERR; i.e., whether it is a soft ISAM or hard AlphaBASIC error.

6.4.2 Using ISAM with a Single Key

The following examples show the correct use of ISAM files that have only a single index file. Note that the procedures show below have always been required; however, it is only with the release of LOKSER that they are being enforced. Thus, some programs which may have seemed to work correctly in the past will now generate errors when LOKSER is installed on your system.

6.4.2.1 Example 1 - Creating a Record

This example shows the proper techique for creating a new record in an ISAM file.

```
20      MAP1  DATA'MAP
30              MAP2  SYM'KEY,S,16       ! Primary key
40                                       ! Open the file for Shared use
50      OPEN #FILE'NO, FILE'NAME, INDEXED, REC'SIZE, REC'NUM
60   NEXT'RECORD:
         { Create the record }

100     ISAM #FILE'NO, 1, SYM'KEY        ! Is it there already?
```

```
110     IF ERF(FILE'NO) <> 33 &
           THEN GOTO ISAM'ERROR          ! If record exists, try again.
120  ADD'RECORD:
130     ISAM #FILE'NO, 5, SYM'KEY        ! Find next free record
140     IF ERF(FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Check for error
150     WRITEL #FILE'NO, DATA'MAP        ! Output the record
160     ISAM #FILE'NO, 3, SYM'KEY        ! Add key to index
170     IF ERF(FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Check for error
180     GOTO NEXT'RECORD                 ! Get another record
190 ISAM'ERROR:                          ! Release record and
200     UNLOKR #FILE'NO                  !   primary index file
210     GOTO NEXT'RECORD
```

6.4.2.2 Example 2 - Deleting a Record

This example shows the proper technique for deleting a record from an ISAM file. The main difference between this technique and the technique used by previous ISAM releases is that you must use a READL statement to lock the data record being deleted.

```
10      MAP1  DATA'MAP
20              MAP2  SYM'KEY,S,16       ! Primary key
30              MAP2  RECORD'DATA,S,48
40                                       ! Open the file for
50                                       ! Shared use:
60      OPEN #FILE'NO, FILE'NAME, INDEXED, REC'SIZE, REC'NUM
70  NEXT'RECORD:
        { Select record to be deleted }

100  LOOKUP'RECORD:
110     ISAM #FILE'NO, 1, SYM'KEY        ! Is it there already?
120     IF ERF(FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! If record does not exist,
130                                      !    try again
140     READL #FILE'NO, DATA'MAP         ! Read and lock the record.
150     ISAM #FILE'NO, 4, SYM'KEY        ! Delete key from index
160     IF ERF(FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Check for error
170     ISAM #FILE'NO, 6, SYM'KEY        ! Delete data record
180     IF ERF(FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Check for error
190     GOTO NEXT'RECORD                 ! Get another record
200  ISAM'ERROR:
210     UNLOKR #FILE'NO                  ! Release record and primary
220     GOTO NEXT'RECORD                 ! index file
```

6.4.2.3 Example 3 - Updating a Record

This example shows the proper technique for updating a record in an ISAM file. Note that if you are changing the key, you MUST do so by first deleting the record (as described in Section 6.4.2.2) and then adding the modified record back (as described in Section 6.4.2.1). You must use READL to read and lock the data record. READL prohibits anyone from changing the record by locking the disk block that contains the data record. You should try to minimize the amount of processing between the READL and WRITE statements (WRITE unlocks the record and the index file).

```
10      MAP1  DATA'MAP
20              MAP2  SYM'KEY,S,16       ! Primary key
30                                       ! Open the file for
35                                       !   Exclusive use.
40      OPEN #FILE'NO, FILE'NAME, INDEXED'EXCLUSIVE, &
                REC'SIZE, REC'NUM
50   NEXT'RECORD:
         { Select record }

100     ISAM #FILE'NO, 1, SYM'KEY        ! Is it there already?
110     IF ERF(FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Record not exist? - try again
120     READL #FILE'NO, DATA'MAP         ! Read and lock the record.

        { Process data in record }

200     WRITE #FILE'NO, DATA'MAP         ! Output and unlock the record
210     GOTO NEXT'RECORD
220  ISAM'ERROR:                         ! Release record and primary
230     UNLOKR #FILE'NO                  !   index file
240     GOTO NEXT'RECORD
```

6.4.3 Using ISAM with Multiple Keys

The only change from pre-LOKSER releases for using multiple keys (i.e., secondary index files) is that you must ensure that the primary index file is locked. If you locate the record using the primary index file, ISAM will lock the index for you automatically (as with single key accesses). However, if you locate a record based on a secondary key, you must use the LOCK statement to explicitly lock the primary index file. The following examples show the correct use of ISAM files with multiple keys. Note that the procedures shown below have always been required. However, it is only with the release of LOKSER that these procedures are being enforced. Thus, some programs which may have seemed to work in the past will generate errors when run on a system with LOKSER installed.

6.4.3.1 Example 1 - Creating a Record

Note that the example shows the secondary key being added before the primary key; this keeps the primary .IDX index file locked until the entire process is completed.

```
10      MAP1 DATA'MAP
20              MAP2  SYM'KEY,S,16      ! Primary key
30              MAP2  SEC'KEY,S,16      ! Secondary key
40                                      ! Open the primary index
50                                      !   file for Shared use
60      OPEN #FILE'NO, FILE'NAME, INDEXED, REC'SIZE, REC'NUM
70                                      ! Open the secondary index
80                                      !   file for Shared use
90      OPEN #SEC'FILE'NO, SEC'FILE'NAME, INDEXED, REC'SIZE, REC'NUM
100  NEXT'RECORD:
        { Create record }

200     ISAM #FILE'NO, 1, SYM'KEY       ! Is the record there already?
210     IF ERF(FILE'NO) <> 33 &
           THEN GOTO ISAM'ERROR         ! Record exists?-process error
220     ISAM #SEC'FILE'NO, 1, SEC'KEY   ! Record in secondary index?
230     IF ERF(SEC'FILE'NO) <> 33 &
           THEN GOTO ISAM'ERROR         ! Record exists?-process error
240  ADD'RECORD:
250     ISAM #FILE'NO, 5, SYM'KEY       ! Find next free record
260     IF ERF(FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR         ! Check for error
270     WRITEL #FILE'NO, DATA'MAP       ! Output the record
280     ISAM #SEC'FILE'NO, 3, SEC'KEY   ! Add the secondary key
290     IF ERF(SEC'FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR         ! Check for error
300     ISAM #FILE'NO, 3, SYM'KEY       ! Add the primary key
310     IF ERF(FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR         ! Check for error
320     GOTO NEXT'RECORD                ! Get the next record
330  ISAM'ERROR:
340     UNLOKR #FILE'NO                 ! Release the record and
350     GOTO NEXT'RECORD                !   the primary index file
```

6.4.3.2 Example 2 - Deleting a Record

This example shows the proper technique for deleting a record from an ISAM file with multiple index files. If you originally locate the record (via an ISAM 1 statement) based on the primary key, the technique shown in Section 5.4.2.2 should be used with an extra step to delete the secondary key. The example below shows the processing that is required if you originally locate the record on a secondary key.

```
10      MAP1 DATA'MAP
20              MAP2  SYM'KEY,S,16       ! Primary key
30              MAP2  SEC'KEY,S,16       ! Secondary key
40                                       ! Open the primary index file
50                                       !   for Exclusive use
60      OPEN #FILE'NO, FILE'NAME, INDEXED'EXCLUSIVE, REC'SIZE, REC'NUM
70                                       ! Open the secondary index file
80                                       !   for Exclusive use
90      OPEN #SEC'FILE'NO, SEC'FILE'NAME, INDEXED'EXCLUSIVE, &
                REC'SIZE, REC'NUM
100  NEXT'RECORD:
        { Select record to delete }

200     LOCK #FILE'NO                    ! Lock primary index file
210     ISAM #SEC'FILE'NO, 1, SEC'KEY    ! Is the record there?
220     IF ERF(SEC'FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          !   No - process error
230  DELETE'RECORD:
240     READL #FILE'NO, DATA'MAP         !   Yes - read and lock record
250     ISAM #SEC'FILE'NO, 4, SEC'KEY    ! Delete key from secondary
255                                      !  index.
260     IF ERF(SEC'FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Check for error
270     ISAM #FILE'NO, 4, SYM'KEY        ! Delete key from primary index
280     IF ERF(FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Check for error
290     ISAM #FILE'NO, 6, SYM'KEY        ! Delete data record
300     IF ERF(FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Check for error
310     GOTO NEXT'RECORD                 ! Get another record
320  ISAM'ERROR:
330     UNLOKR #FILE'NO                  ! Release the record and the
340     GOTO NEXT'RECORD                 !   primary index file
```

6.4.3.3 Example 3 - Updating a Record

Remember that if the modifications to the record included changing the primary key, you must first delete the old record (as discussed in Section 6.4.3.2) and then add the new record (Section 6.4.3.1).

```
10      MAP1  DATA'MAP
20              MAP2  SYM'KEY,S,16       ! Primary key
30              MAP2  SEC'KEY,S,16       ! Secondary key
40                                       ! Open the primary index file
50                                       !   for Shared use
60      OPEN #FILE'NO, FILE'NAME, INDEXED, REC'SIZE, REC'NUM
70                                       ! Open the secondary index file
80                                       !   for Shared use
90      OPEN #SEC'FILE'NO, SEC'FILE'NAME, INDEXED, &
             REC'SIZE, REC'NUM
```

```
100  NEXT'RECORD:
        { Select a record for updating }

200     LOCK #FILE'NO                    ! Lock primary index file.
210     ISAM #SEC'FILE'NO, 1, SEC'KEY    ! Find the record
220     IF ERF(SEC'FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Is record there?
230     READL #FILE'NO, DATA'MAP         !   Yes - get and lock it.
240     OLD'SEC'KEY = SEC'KEY
        { Update data in record }

300     IF OLD'SEC'KEY = SEC'KEY  &
           THEN GOTO OUT'RECORD          ! Did secondary key change?
310                                      ! Secondary key changed --
320                                      !   remove old index entry
330     ISAM #SEC'FILE'NO, 4, OLD'SEC'KEY
340     IF ERF(SEC'FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Check for error
350     ISAM #SEC'FILE'NO, 3, SEC'KEY    ! Add new key
360     IF ERF(SEC'FILE'NO) <> 0  &
           THEN GOTO ISAM'ERROR          ! Check for error
370  OUT'RECORD:                         ! Output and unlock the record
380     WRITE #FILE'NO, DATA'MAP
390     GOTO NEXT'RECORD
400  ISAM'ERROR:
410     UNLOKR #FILE'NO                  ! Release the record and
420     GOTO NEXT'RECORD                 !  unlock primary index file.
```

CHAPTER 7

WRITING AMOS ASSEMBLY LANGUAGE PROGRAMS THAT ACCESS ISAM

NOTE: This section assumes that you are an experienced assembly language programmer and that you are familiar with the Alpha Micro CPU instruction set and the AMOS monitor calls. For information on these topics, refer to the AMOS Assembly Language Programmer's Manual, the WD16 Microcomputer Programmer's Reference Manual, DWM-00100-04, and the AMOS Monitor Calls Manual.

The ISAM program is implemented as a FETCHable memory module which allows the assembly language programmer easy access to the features of indexed sequential files. (NOTE: FETCH is an AMOS monitor call. Refer to the AMOS Monitor Calls Manual for information on the routines within the operating system (called "monitor calls") that have been made available to your assembly language programs.) It is through the ISAM module that high level languages such as AlphaBASIC gain access to indexed sequential files. The ISAM program is fully re-entrant, and could therefore be made resident in system memory if more than one user at a time is going to be using indexed sequential files.

The ISAM program itself takes up approximately 4K bytes of memory. In addition to this space, another 1 to 4K bytes is required for each indexed sequential file that you are processing. This memory space is usually allocated by the ISAM system using the GETMEM monitor call; you may, however, allocate your own buffer areas (see Section 7.2.2).

7.1 GENERAL CALLING SEQUENCE

The various ISAM subroutines are called via a dispatch table at the start of the ISAM program. To make things easier, the file DSK0:ISUSYM.MAC[7,7] defines the table offsets. This file also contains symbols for the various return codes. All table offsets begin with a period (e.g., .ICLOS, the close routine). All return codes have the general form IS.xxx (e.g., IS.EOF, the end-of-file return code).

NOTE: ISUSYM.MAC is designed to be COPYed by your assembly language program.

To call the close routine (.ICLOS) with the base of the ISAM.SYS module contained in register R4, use the following code:

```
CALL    .ICLOS(R4)
```

All arguments are passed in registers. NOTE: As of AMOS Versions 5.0 and later, the contents of R0 are significant when making some of the calls discussed below-- see the call descriptions for more information. For information on the flags you can set in R0 for some of the calls discussed below, see Chapter 9, "Standard ISAM Symbols for Assembly Language Programmers."

Each call returns with a completion code in R0. A successful return (IS.SUC) is indicated by a zero in R0; the indicators (also known as condition codes or condition flags) on return reflect success or error status. The Z-bit is set if successful (BNE branches on error).

7.2 INITIALIZING THE ISAM SYSTEM (.INIT)

Before your program can access an indexed sequential file, you must tell the ISAM system that you exist; this is done via the .INIT call. The .INIT call allocates space for the user's impure variables and does minor housekeeping chores. NOTE: Your program calls .INIT only once regardless of the number of ISAM files that are to be opened.

Calling Sequence:

Parameters:	R2	User allocation routine address (optional)
	R3	User deallocation routine address (optional)
	R4, R5	Used to pass information to user memory allocation routines (optional).
	CALL	.INIT(Rn)
Returns:	R0	Completion code
	R5	User memory pointer
	Indicators	Z if no error

The user memory pointer that is returned in R5 is a pointer to your impure area. This pointer is needed by all other ISAM calls; if convenient, leave it in R5 since all calls look for it there.

7.2.1 User Supplied Allocation and Deallocation Routines

In many cases, the program calling the ISAM program will do its own memory management, and not want ISAM to use GETMEMs to do so. To allow you to do your own allocation, the .INIT call allows the passing of allocation and deallocation routine addresses. .INIT uses its own routines (which use GETMEMs) if you pass a zero instead of an address.

The user allocation routine is called with the desired module size in R1. The .INIT call expects the address of the assigned module to be returned in R1. You may not modify any other registers.

The user deallocation routine is called with the address of the module to be deleted in R1. Do not modify any other registers. If you pass a zero to .INIT in R3, no deallocation occurs.

The current version of ISAM allows you to move any of the modules that ISAM requests as well as the ISAM program itself. ISAM is immune to such movement as long as the user memory pointer (in R5) and the FPN (file pair number, see Section 7.6.1) associated with a given file are updated to show any movement. (The FPN is usually in R1.)

7.3 FINALIZING ISAM PROCESSING (.IFIN)

When you are through processing indexed sequential files, you must call the .IFIN routine. This call deallocates any space used by ISAM if a user deallocation routine has been provided; otherwise the modules are not deleted until the job EXITs. (EXIT is an AMOS monitor call.)

Calling Sequence:

```
Parameters:        R5        User memory pointer

                   CALL      .IFIN(Rn)

Returns:           R0        IS.SUC
            Indicators       Z if no error
```

The .IFIN routine cannot fail, therefore it always returns the successful completion code in R0.

7.4 OPENING A FILE FOR PROCESSING (.IOPNR)

You must open an indexed sequential file via this call before you can process the file in any way. Also use this call when opening a secondary index file for processing a previously opened data file. If you execute this call on a primary index file, the call also opens the associated data file; if you execute the call on a secondary index file, the call opens the index file only. Thus to process a data file with a secondary index file, you must execute two .IOPNR calls: once to open the data file and primary index file, and once to open the secondary index file.

Calling Sequence:

Parameters:	R0	You may set the following flags: IS.EXC - Open in Exclusive mode; otherwise Counted Update mode is used. IS.ERP - Print error messages before returning. IS.FWT - Wait for file if it is in use.
	R2	Pointer to ASCII filespec string describing the index file to be opened. If the index file is a primary index, the data file must have the same name.
	R5	User memory pointer
	CALL	.IOPNR(Rn)
Returns:	R0	Completion code
	R1	Unique File Pair Number (FPN)
	Indicators	Z if no error

The file pair number (FPN) is a pointer to the memory module that has been allocated for the storage needed by this particular indexed sequential file. The read, write, and delete routines use the FPN to tell the ISAM program which indexed sequential files to process of the ones you may have open.

If you must move the module allocated by .IOPNR, you may do so as long as you also update the FPN.

7.5 CLOSING THE FILE AFTER PROCESSING (.ICLOS)

The .ICLOS call to ISAM performs one of three separate functions. The standard call is used to inform ISAM that you are finished with the file. In this mode, ISAM performs the necessary housekeeping functions and deallocates any memory used by the file if a user deallocation routine has not been provided. The other two modes are used to lock and unlock the primary index file. (See the definition of the R0 flags, below.)

Calling Sequence:

	R0	You may set the following flags: IS.LOK - Lock IS.UNL - Unlock (If both are set, performs lock; if both are zero, just closes.)
Parameters:	R1	File pair number (FPN)
	R5	User memory pointer
	CALL	.ICLOS(Rn)
Returns:	R0	Completion code
	Indicators	Z if no error

The file pair number used in R1 is that value returned by .IOPNR.

7.6 LOCATING A FREE DATA RECORD (.IGTFR)

Use this call to get the relative record number of the next available data record in the data file. The primary index file must already be locked when you make this call.

Calling Sequence:

Parameters:	R1	File pair number (FPN) of primary index file
	R5	User memory pointer
	CALL	.IGTFR(Rn)
Returns:	R0	Completion code
	R1	Low-order relative record number of the data record
	R2	High-order relative record number of the data record
	Indicators	Z if no error

The FPN supplied in R1 must refer to the primary index file associated with the data file from which a free record is to be obtained.

7.7 DELETING A DATA RECORD (.IDLFR)

Use this call to return a data record to the free record list. The primary index file must already be locked when you make this call. The .IDLFR call unlocks the primary index file at the completion of the call.

Calling Sequence:

Parameters:	R1	File pair number (FPN) of primary index file
	R2	Low-order relative record number of the data record
	R3	High-order relative record number of the data record
	R5	User memory pointer
	CALL	.IDLFR(Rn)
Returns:	R0	Completion code
	Indicators	Z if no error

The FPN supplied in R1 must refer to a primary index file.

7.8 READING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLRD)

Use this call to read the data record pointed to by the relative record number. The primary index file must already be locked when you make this call. If the Exclusive flag is not set (i.e., INPUT was used, not INPUTL), this call unlocks the index file.

Calling Sequence:

Parameters:

	R0	You may set the following flags: IS.REX - Read Exclusive IS.RWT - Wait for record if it is in use.
	R1	Low-order relative record number of the data record
	R2	High-order relative record number of the data record
	R3	File pair number (FPN)
	R4	Buffer address
	R5	User memory pointer
	CALL	.IRLRD(Rn)
Returns:	R0	Completion code
	Indicators	Z if no error

The FPN supplied in R3 must refer to the primary index file associated with the data file.

7.9 WRITING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLWT)

Use this call to write or update the data record pointed to by the relative record number. The primary index file must already be locked when you make this call. If the Exclusive flag is not set (an OUTPUT has been used instead of an OUTPTL), the call unlocks the index file.

Calling Sequence:

Parameters:	R0	You may set the following flags: IS.REX - Lock and write IS.RWT - Wait for record if it is in use.
	R1	Low-order relative record number of the data record
	R2	High-order relative record number of the data record
	R3	File pair number (FPN)
	R4	Buffer address
	R5	User memory pointer
	CALL	.IRLWT(Rn)
Returns:	R0	Completion code
	Indicators	Z if no error

7.10 FINDING A RECORD (.IREDR)

Use this call to use a symbolic key to find the relative record number of a data record. If the index file is a primary index file, this call locks it.

Calling Sequence:

Parameters:	R1	File pair number (FPN) of the desired index file
	R3	Pointer to symbolic key
	R5	User memory pointer
	CALL	.IREDR(Rn)
Returns:	R0	Completion code
	R1	Low-order relative record number of data record
	R2	High-order relative record number of data record
	Indicators	Z if no error

The FPN supplied in R1 may refer to any open index file.

7.11 ADDING A SYMBOLIC KEY (.IWRTR)

Use this call to add a key entry to an index file given a user supplied data record number. The primary index file must already be locked when you make this call. This call unlocks the primary index file.

Calling Sequence:

Parameters:	R1	File pair number (FPN) of desired index file
	R2	Pointer to symbolic key
	R3	Low-order relative record number of the data record
	R4	High-order relative record number of the data record
	R5	User memory pointer
	CALL	.IWRTR(Rn)
Returns:	R0	Completion code
	Indicators	Z if no error

The FPN supplied in R1 may refer to any open index file. The relative record number in R3 and R4 will usually be a record number returned by the .IGTFR call.

7.12 DELETING A KEY (.IDELK)

Use this call to delete a key from an index file. The primary index file must already be locked when you make this call.

Calling Sequence:

Parameters:	R1	File pair number (FPN) of desired index file
	R3	Pointer to symbolic key
	R5	User memory pointer
	CALL	.IDELK(Rn)
Returns:	R0	Completion code
	R1	Low-order relative record number of deleted key
	R2	High-order relative record number of deleted key
	Indicators	Z if no error

The relative record number in R1 and R2 refers to the data record associated with the deleted key within the index file referred to by the supplied FPN. The data record is not deleted in the data file; do this by using the .IDLFR call when you are sure that there are no keys left in the index file that refer to that data record.

7.13 FINDING THE NEXT SEQUENTIAL KEY (.SREDR)

When printing reports or posting data, it is often useful to be able to go through the records in the indexed sequential file in ascending key order. The .SREDR call makes this possible; it returns the relative record number of the record that immediately follows the one returned by the last call to ISAM. It thus makes it possible to start sequential processing by key anywhere in the file. To do so, use the .IREDR call to get the first key you wish to use. Then call .SREDR to get the key following the one read by .IREDR. You can get the next key by doing another .SREDR, ad infinitum. If .IREDR does not find the key specified, the following .SREDR returns the record with the key closest to (but greater than) the one not found. Thus, to read the file from the very beginning, try to do a .IREDR with a key of zero. This call will almost always fail, but the following .SREDR grabs the very first record in the file.

If the file contains a key of all zero, the initial .IREDR will succeed and that record should be processed as the first record before doing any .SREDRs. When the file is initially opened by .IOPNR, it is set up so that .SREDR gets the first key, unless the first key is all zero. Therefore it is almost always possible to open the file and read it sequentially. Since keys are expected to be ASCII, and an all-null key is not very sensible, the various utilities assume that the first key is not zero. When the end of the file is reached by the last .SREDR, the end-of-file (IS.EOF) completion code is returned.

This call locks the index file if it is not the primary index file.

Calling Sequence:

Parameters:	R1	File pair number (FPN) of desired index
	R3	Contains 0 if symbolic key is not to be returned; otherwise, R3 contains a pointer to the location in memory that the returned symbolic key is to be placed in.
	R5	User memory pointer
	CALL	.SREDR(Rn)
Returns:	R0	Completion code
	R1	Low-order relative record number of data record
	R2	High-order relative record number of the data record
	Indicators	Z if no error

NOTE: If you do not want the symbolic key returned, be sure that R3 is cleared to zero before your program calls .SREDR. Remember that if R3 is nonzero, ISAM will return the symbolic key to the memory location specified by the contents of R3-- if R3 happens to contain invalid data, ISAM may damage important data in memory.

CHAPTER 8

WRITING AMOS/L ASSEMBLY LANGUAGE PROGRAMS THAT ACCESS ISAM

NOTE: This section assumes that you are an experienced assembly language programmer and that you are familiar with the Alpha Micro AM-100/L CPU instruction set and the AMOS/L monitor calls. For information on these topics, refer to the AMOS/L Assembly Language Programmer's Manual, DSS-10007-00, the AM-100/L Instruction Set, DSS-10006-00, and the AMOS/L Monitor Calls, DSS-10003-00.

The ISAM program is implemented as a FETCHable memory module which allows the assembly language programmer easy access to the features of indexed sequential files. (NOTE: FETCH is an AMOS/L monitor call. Refer to the AMOS/L Monitor Calls for information on the routines within the operating system (called "monitor calls") that have been made available to your assembly language programs.) It is through the ISAM module that high level languages such as AlphaBASIC gain access to indexed sequential files. The ISAM program is fully re-entrant, and could therefore be made resident in system memory if more than one user at a time is going to be using indexed sequential files.

The ISAM program itself takes up approximately 4K bytes of memory. In addition to this space, another 1 to 4K bytes is required for each indexed sequential file that you are processing. This memory space is usually allocated by the ISAM system using the GETMEM monitor call; you may, however, allocate your own buffer areas (see Section 8.2.2).

8.1 GENERAL CALLING SEQUENCE

The various ISAM subroutines are called via a dispatch table at the start of the ISAM program. To make things easier, the file ISUSYM.M68 defines the table offsets. This file also contains symbols for the various return codes. All table offsets begin with a period (e.g., .ICLOS, the close routine). All return codes have the general form IS.xxx (e.g., IS.EOF, the end-of-file return code).

NOTE: ISUSYM.M68 is designed to be COPYed by your assembly language program.

To call the close routine (.ICLOS) with the base of the ISAM.SYS module contained in register A4, use the following code:

```
CALL    .ICLOS(A4)
```

All arguments are passed in registers. NOTE: As of AMOS/L Versions 1.1 and later, the contents of D0 are significant when making some of the calls discussed below-- see the call descriptions for more information. For information on the flags you can set in D0 for some of the calls discussed below, see Chapter 9, "Standard ISAM Symbols for Assembly Language Programmers."

Each call returns with a completion code in D0. A successful return (IS.SUC) is indicated by a zero in D0; the indicators (also known as condition codes or condition flags) on return reflect success or error status. The Z-bit is set if successful (BNE branches on error).

8.2 INITIALIZING THE ISAM SYSTEM (.INIT)

Before your program can access an indexed sequential file, you must tell the ISAM system that you exist; this is done via the .INIT call. The .INIT call allocates space for the user's impure variables and does minor housekeeping chores. NOTE: Your program calls .INIT only once regardless of the number of ISAM files that are to be opened.

Calling Sequence:

Parameters:	A2	User allocation routine address (optional)
	A3	User deallocation routine address (optional)
	A4, A5	Used to pass information to user memory allocation routines (optional).
	CALL	.INIT(An)
Returns:	D0	Completion code
	A5	User memory pointer
	Indicators	Z if no error

The user memory pointer that is returned in A5 is a pointer to your impure area. This pointer is needed by all other ISAM calls; if convenient, leave it in A5 since all calls look for it there.

8.2.1 User Supplied Allocation and Deallocation Routines

In many cases, the program calling the ISAM program will do its own memory management, and not want ISAM to use GETMEMs to do so. To allow you to do your own allocation, the .INIT call allows the passing of allocation and deallocation routine addresses. .INIT uses its own routines (which use GETMEMs) if you pass a zero instead of an address.

The user allocation routine is called with the desired module size in D1. The .INIT call expects the address of the assigned module to be returned in A1. You may not modify any other registers.

The user deallocation routine is called with the address of the module to be deleted in A1. Do not modify any other registers. If you pass a zero to .INIT in A3, no deallocation occurs.

The current version of ISAM allows you to move any of the modules that ISAM requests as well as the ISAM program itself. ISAM is immune to such movement as long as the user memory pointer (in A5) and the FPN (file pair number, see Section 8.6.1) associated with a given file are updated to show any movement. (The FPN is usually in A1.)

8.3 FINALIZING ISAM PROCESSING (.IFIN)

When you are through processing indexed sequential files, you must call the .IFIN routine. This call deallocates any space used by ISAM if a user deallocation routine has been provided; otherwise the modules are not deleted until the job EXITs. (EXIT is an AMOS/L monitor call.)

Calling Sequence:

```
Parameters:        A5       User memory pointer

                   CALL     .IFIN(An)

Returns:           D0       IS.SUC
            Indicators      Z if no error
```

The .IFIN routine cannot fail, therefore it always returns the successful completion code in D0.

8.4 OPENING A FILE FOR PROCESSING (.IOPNR)

You must open an indexed sequential file via this call before you can process the file in any way. Also use this call when opening a secondary index file for processing a previously opened data file. If you execute this call on a primary index file, the call also opens the associated data file; if you execute the call on a secondary index file, the call opens the index file only. Thus to process a data file with a secondary index file, you must execute two .IOPNR calls: once to open the data file and primary index file, and once to open the secondary index file.

Calling Sequence:

Parameters:	D0	You may set the following flags: IS.EXC - Open in Exclusive mode; otherwise Counted Update mode is used. IS.ERP - Print error messages before returning. IS.FWT - Wait for file if it is in use.
	A2	Pointer to ASCII filespec string describing the index file to be opened. If the index file is a primary index, the data file must have the same name.
	A5	User memory pointer
	CALL	.IOPNR(An)
Returns:	D0	Completion code
	A1	Unique File Pair Number (FPN)
	Indicators	Z if no error

The file pair number (FPN) is a pointer to the memory module that has been allocated for the storage needed by this particular indexed sequential file. The read, write, and delete routines use the FPN to tell the ISAM program which indexed sequential files to process of the ones you may have open.

If you must move the module allocated by .IOPNR, you may do so as long as you also update the FPN.

8.5 CLOSING THE FILE AFTER PROCESSING (.ICLOS)

The .ICLOS call to ISAM performs one of three separate functions. The standard call is used to inform ISAM that you are finished with the file; in this mode, ISAM performs the necessary housekeeping functions and deallocates any memory used by the file if a user deallocation routine has been provided. The other two modes are used to lock and unlock the primary index file.

Calling Sequence:

```
Parameters:
                    D0      You may se the following flags:
                              IS.LOK - Lock
                              IS.UNL - Unlock
                            (If both are set, performs lock;
                            if both are zero, just closes.)
                    A1      File pair number (FPN)
                    A5      User memory pointer

                    CALL    .ICLOS(An)

Returns:            D0      Completion code
            Indicators      Z if no error
```

The file pair number used in A1 is that value returned by .IOPNR.

8.6 LOCATING A FREE DATA RECORD (.IGTFR)

Use this call to get the relative record number of the next available data record in the data file. The primary index file must already be locked when you make this call.

Calling Sequence:

```
Parameters:         A1      File pair number (FPN) of primary index file
                    A5      User memory pointer

                    CALL    .IGTFR(An)

Returns:            D0      Completion code
                    D1      Relative record number of the data record
            Indicators      Z if no error
```

The FPN supplied in A1 must refer to the primary index file associated with the data file from which a free record is to be obtained.

8.7 DELETING A DATA RECORD (.IDLFR)

Use this call to return a data record to the free record list. The primary index file must already be locked when you make this call. This call unocks the primary index file at the completion of the call.

Calling Sequence:

```
Parameters:        A1      File pair number (FPN) of primary index file
                   D1      Relative record number of the data record
                   A5      User memory pointer

                   CALL    .IDLFR(An)

Returns:           D0      Completion code
             Indicators    Z if no error
```

The FPN supplied in A1 must refer to a primary index file.

8.8 READING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLRD)

Use this call to read the data record pointed to by the relative record number. The primary index file must already be locked when you make this call. If the Exclusive flag is not set (that is, an INPUT was used instead of an INPUTL), the call unlocks the primary index file.

Calling Sequence:

```
Parameters:
                   D0      You may set the following flags:
                            IS.REX - Read Exclusive
                            IS.RWT - Wait for record if it is in use.
                   D1      Relative record number of the data record
                   A3      File pair number (FPN)
                   A4      Buffer address
                   A5      User memory pointer
                   CALL    .IRLRD(An)

Returns:           D0      Completion code
             Indicators    Z if no error
```

The FPN supplied in A3 must refer to the primary index file associated with the data file.

8.9 WRITING A DATA RECORD BY RELATIVE RECORD NUMBER (.IRLWT)

Use this call to write or update the data record pointed to by the relative record number. The primary index file must already be locked when you make this call. If the Exclusive flag is not set (an OUTPUT was done instead of an OUTPTL), the call unlocks the index.

Calling Sequence:

Parameters:

D0	You may set the following flags:
	IS.REX - Lock and write
	IS.RWT - Wait for record if it is in use.
D1	Relative record number of the data record
A3	File pair number (FPN)
A4	Buffer address
A5	User memory pointer
CALL	.IRLWT(An)

Returns:

D0	Completion code
Indicators	Z if no error

8.10 FINDING A RECORD (.IREDR)

Use this call to use a symbolic key to find the relative record number of a data record. If the index file is a primary index file, this call locks it.

Calling Sequence:

Parameters:

A1	File pair number (FPN) of the desired index file
A3	Pointer to symbolic key
A5	User memory pointer
CALL	.IREDR(An)

Returns:

D0	Completion code
D1	Relative record number of data record
Indicators	Z if no error

The FPN supplied in A1 may refer to any open index file.

8.11 ADDING A SYMBOLIC KEY (.IWRTR)

Use this call to add a key entry to an index file given a user supplied data record number. The primary index file must already be locked when you make this call. This call unlocks the primary index file when it is done.

Calling Sequence:

Parameters:	A1	File pair number (FPN) of desired index file
	A2	Pointer to symbolic key
	D3	Relative record number of the data record
	A5	User memory pointer
	CALL	.IWRTR(An)
Returns:	D0	Completion code
	Indicators	Z if no error

The FPN supplied in A1 may refer to any open index file. The relative record number in D will usually be a record number returned by the .IGTFR call.

8.12 DELETING A KEY (.IDELK)

Use this call to delete a key from an index file. The primary index file must already be locked when you make this call.

Calling Sequence:

Parameters:	A1	File pair number (FPN) of desired index file
	A3	Pointer to symbolic key
	A5	User memory pointer
	CALL	.IDELK(An)
Returns:	D0	Completion code
	D1	Relative record number of deleted key
	Indicators	Z if no error

The relative record number in D1 refers to the data record associated with the deleted key within the index file referred to by the supplied FPN. The data record is not deleted in the data file; do this by using the .IDLFR call when you are sure that there are no keys left in the index file that refer to that data record.

8.13 FINDING THE NEXT SEQUENTIAL KEY (.SREDR)

When printing reports or posting data, it is often useful to be able to go through the records in the indexed sequential file in ascending key order. The .SREDR call makes this possible; it returns the relative record number of the record that immediately follows the one returned by the last call to ISAM. It thus makes it possible to start sequential processing by key anywhere in the file. To do so, use the .IREDR call to get the first key you wish to use. Then call .SREDR to get the key following the one read by .IREDR. You can get the next key by doing another .SREDR, ad infinitum. If .IREDR does not find the key specified, the following .SREDR returns the record with the key closest to (but greater than) the one not found. Thus, to read the file from the very beginning, try to do a .IREDR with a key of zero. This call will almost always fail, but the following .SREDR grabs the very first record in the file.

If the file contains a key of all zero, the initial .IREDR will succeed and that record should be processed as the first record before doing any .SREDRs. When the file is initially opened by .IOPNR, it is set up so that .SREDR gets the first key, unless the first key is all zero. Therefore it is almost always possible to open the file and read it sequentially. Since keys are expected to be ASCII, and an all-null key is not very sensible, the various utilities assume that the first key is not zero. When the end of the file is reached by the last .SREDR, the end-of-file (IS.EOF) completion code is returned.

If the index file is the primary index file, this call locks it.

Calling Sequence:

Parameters:	A1	File pair number (FPN) of desired index
	A3	Contains 0 if symbolic key is not to be returned; otherwise, A3 contains a pointer to the location in memory that the returned symbolic key is to be placed in.
	A5	User memory pointer
	CALL	.SREDR(An)
Returns:	D0	Completion code
	D1	Relative record number of data record
	Indicators	Z if no error

NOTE: If you do not want the symbolic key returned, be sure that A3 is cleared to zero before your program calls .SREDR. Remember that if A3 is nonzero, ISAM will return the symbolic key to the memory location specified by the contents of A3-- if A3 happens to contain invalid data, ISAM may damage important data in memory.

CHAPTER 9

STANDARD ISAM SYMBOLS FOR ASSEMBLY LANGUAGE PROGRAMMERS

We have provided a symbols file (ISUSYM.MAC on AMOS systems; ISUSYM.M68 on AMOS/L systems) to make life easier for the assembly language programmer. This chapter describes the contents of that file. We have broken this information into three sections: 1. those offsets used when invoking ISAM; 2. the flags you can set when making the calls; and, 3. the completion codes returned by the various ISAM functions.

9.1 CALLING SYMBOLS

The following symbols define entry offsets in the ISAM package. See Sections 7.1 and 8.1 of this manual for more information.

.INIT	Call the initialization routine
.IFIN	Call the finalization routine
.IOPNR	Call the file open routine
.ICLOS	Call the file close routine
.IGTFR	Call the get free record routine.
.IDLFR	Call the data record deletion routine.
.IRLRD	Call the read data record via relative record number routine.
.IRLWT	Call the write data record via relative record number routine.
.IREDR	Call the read returning relative record number routine.
.IWRTR	Call the write using relative record number routine.
.IDELK	Call the key deletion routine.
.SREDR	Call the read sequential relative record number routine.

9.2 CALLING FLAGS

The following symbols are the flags you can set in R0 (on AMOS systems) or D0 (on AMOS/L systems) when making some of the ISAM calls discussed in Chapters 7 and 8. Always use these symbols rather than their values in ISUSYM.MAC or ISUSYM.M68 since those values could change in the future.

IS.EXC	(10 octal)	Open Directory for Exclusive access.
IS.ERP	(1000 octal)	Print any error messages (only for file open calls).
IS.LOK	(2000 octal)	Lock index file (only for file close calls).
IS.UNL	(4000 octal)	Unlock index file (only for file close calls).
IS.REX	(10000 octal)	Read record Exclusively (for read or write operations).
IS.RWT	(40000 octal)	Wait for record if it is in use.
IS.FWT	(100000 octal)	Wait for file if it is in use.

9.3 COMPLETION CODE SYMBOLS

The following symbols name the completion codes returned in R0 (on AMOS systems) or D0 (on AMOS/L systems) upon completion of an ISAM call. Always use these symbols rather than their values in ISUSYM.MAC or ISUSYM.M68, since those values could change in the future.

IS.SUC	The successful completion code (will always be zero).
IS.DNR	Parameters supplied do not match those in the Directory Rock.
IS.RNF	Record not found.
IS.DPK	Attempt to add duplicate key.
IS.LSS	Index file link structure is smashed.
IS.XFL	Index file is full.
IS.AFL	Data file is full.
IS.EOF	End of file encountered on sequential read.
IS.ILP	Illegal operation sequence

An error code of 2 (handled by AlphaBASIC as a SYSTEM ERROR; which is why this code was chosen) means either that something in ISAM or the ISAM file structure is in error (as in a bug) or that you used an obsolete ISAM call.

CHAPTER 10

USING ISAM FROM WITHIN ALPHABASIC

The following pages are a brief summary of the AlphaBASIC ISAM commands that your AlphaBASIC programs can use to access ISAM indexed sequential files. (Remember that an indexed sequential file is made up of both an ISAM data file and at least one ISAM index file, all of which are random files.) For more information on AlphaBASIC itself, refer to the AlphaBASIC User's Manual, DWM-00100-01. The following discussions assume that you are already familiar with opening and closing random files, and that you understand the AlphaBASIC READ and WRITE statements. For more information on using files, refer to the AlphaBASIC User's Manual.

For information on the LOKSER File Locking Service System, see Chapter 6 of this manual-- that chapter contains examples of the proper procedures to use when locking ISAM files.

You must use the ISAM utility program ISMBLD to create an indexed sequential file before your AlphaBASIC program can access it. Although no features exist within AlphaBASIC to create an indexed sequential file, your AlphaBASIC program can create and execute a command file that invokes ISMBLD with a list of file parameters. All data files must have an extension of .IDA and all index files must have the .IDX extension.

Before you run your AlphaBASIC program, make sure that ISAM.SYS has been loaded into memory if the System Operator has not arranged to have ISAM.SYS resident in system memory.

10.1 OPENING AN INDEXED SEQUENTIAL FILE

To open an ISAM indexed sequential index file, use the AlphaBASIC OPEN command. Your program must include an OPEN statement that assigns a file channel to the indexed sequential file before the program makes any other references to that file.

This statement takes the same form as the OPEN statement for ordinary random files except that you must specify either INDEXED or INDEXED'EXCLUSIVE mode rather than RANDOM mode. (Remember that your

ISAM files are random data files. For information on using random files, see Chapter 15 of the AlphaBASIC User's Manual.) The OPEN statement takes this form:

```
OPEN #file-channel,filespec,mode,record-size,relative-record-number{,wait}
```

1. #file-channel - specifies the file channel you want to assign to the indexed sequential file. Any numeric expression that evaluates to an integer from 0-65535 (0 is the user terminal). This is the number you reference when using the ISAM, READ, READL, WRITE, and WRITEL statements.

2. Filespec - The filespec is any string expression that evaluates to a legal AMOS file specification. (It may optionally include account and device specifications.) It specifies the name of the indexed sequential file you created using ISMBLD (that is, the data file/primary index file combination you built) or specifies the name of a secondary index file created with ISMBLD. (If the OPEN statement refers to a secondary index file, you must have previously opened the corresponding data file/primary index file on another file channel.)

   Note that the primary index file always has the same name as the data file, but has a .IDX extension; the data file has a .IDA extension.

3. Mode - If you wish ISAM to access the indexed sequential file in Counted Update mode, use the INDEXED keyword as the file mode; if you want ISAM to access the file in Exclusive Open mode, use the keyword INDEXED'EXCLUSIVE. (For information on Counted Update mode and on Exclusive Open mode, see Section 1.3, "ISAM Access Modes.") If the LOKSER File Locking Service System is installed on your system, opening an ISAM file in INDEXED mode will open it for Shared use; opening it in INDEXED'EXCLUSIVE mode will open it for Exclusive use.

4. Record-size - An expression that specifies the logical record size for read/write operations.

5. Relative record number - A floating point variable that will hold the relative record number returned by an ISAM function. (See Section 10.2, "The ISAM Statement.")

6. {,wait} - You may use this optional clause if LOKSER is installed on your system. The clause may take two forms: WAIT'RECORD or WAIT'FILE.

   If you do not use the {,wait} clause, and LOKSER is installed on your system, if you try to access a record in a file or if you try to access a file and the resource you need is already locked, LOKSER will prevent you from accessing it.

What happens then depends on your program-- if your program has error trapping enabled, the actions specified by your error handling routine will be performed; if your program does not have error trapping enabled, your program will be interrupted and the user of your program will return to monitor command level. (For information on dealing with the usual system errors (e.g., "?File not found" or "?Disk not mounted") refer to the AlphaBASIC User's Manual, in particular Chapter 17, which gives information on error trapping.)

If you specify WAIT'FILE, LOKSER will not report an error if the program tries to access a file that is already locked, but will instead put the job using the program to sleep until the file is available.

If you specify WAIT'RECORD, LOKSER will not report an error if the program tries to access a record that is already locked, but will instead put the job using the program to sleep until the record is available.

For an example of the use of the OPEN statement, refer to the sample AlphaBASIC program at the end of this chapter. Below are several sample OPEN statements:

```
220     OPEN #100, "LABELS", INDEXED, RECSIZE, RELKEY1
230     OPEN #200, "HASH", INDEXED, RECSIZE, RELKEY1

100     OPEN #300, "MAIL", INDEXED'EXCLUSIVE, REC'SIZ, &
                REL'KEY, WAIT'FILE
```

The first two program lines above assume that there exists a data file named LABELS.IDA and a primary index file named LABELS.IDX. Line 220 opens that indexed sequential file. Line 230 opens a secondary index file associated with LABELS.IDA. Note that RECSIZE and RELKEY1 are identical for both OPEN statements; this is because both OPEN statements refer to the SAME data file, LABELS.IDA. The RECSIZE and RELKEY1 are used by subsequent READ and WRITE commands to access the data file.

10.2 THE ISAM STATEMENT

The purpose of the ISAM statement is to allow you to use the ISAM program from within your AlphaBASIC program to: 1. find a record in the data file by symbolic key (returning the relative record number in the variable specified by the indexed sequential file OPEN statement); 2. find the next data record (by the order in which the symbolic keys occur in the index file); 3. add a symbolic key to an index file; 4. delete a symbolic key from an index file; 5. locate next free data record in data file (returning relative record number in the variable specified by the appropriate OPEN statement); 6. delete a record from

a data file, and return that record to the free list; and, 7. find the next data record (by the order in which the symbolic keys occur in the index file) and return the symbolic key.

The ISAM statement follows this form:

ISAM #file-channel, code, symbolic-key

1. #file-channel - Specifies the file channel assigned by an OPEN statement to either the data file/primary index file or the secondary index file (depending on which set of symbolic keys you want to access).

2. Code - A numeric value that selects one of the functions mentioned above. May be any legal numeric expression which is resolved at runtime.

3. Symbolic-key - Specifies the symbolic key to be used in locating a data record. You must always specify a symbolic key even if a function does not require the use of one. (This simplifies syntax checking.) If you wish, you may use a dummy string variable in such cases.

10.2.1 ISAM Statement Codes

The ISAM statement can perform seven different functions. You may select one of these functions by supplying the appropriate code number (see below) to the ISAM statement. An error will result if you do not supply a valid code number.

Some of the functions below require a relative record number as input; others return a relative record number to be used when your READ and WRITE statements access the data file. In either case, the ISAM functions pass the relative record number in the variable specified in the OPEN statement for the data file/primary index file. READ and WRITE statements also use that variable for locating the data file record that they are going to access. Remember that the ISAM statement does not directly access the data file. Instead, it gives you the information you need to access the data file yourself using the relative record number returned by ISAM.

CODE 1 - Searches the index file selected by #file-channel for the key that matches the symbolic-key. If it finds a match, it returns the relative record number of the data file record containing that key. If it does not find a match, it returns an error code 33 in ERF(X), where X is the file channel number of the file you want to check. (See Section 10.5, "Error Processing").

CODE 2 - Accesses the index file selected by #file-channel and finds the next symbolic key. Returns the relative record number of the data file record associated with that symbolic key in preparation for a READ or a WRITE to the data file. If this is the first access to the file after the OPEN statement, it finds the first symbolic key in the index file. If this function follows a previous code 1 statement, the function finds the next symbolic key after the code 1 symbolic key. If there are no more keys in the index file, the function returns an end-of-index-file error (38): make no further accesses to the data file until you make another ISAM call that returns a valid relative record number.

CODE 3 - Adds the specified symbolic key to the index file selected by #file-channel. Also adds the relative record number specified by the variable in the OPEN statement. You will usually set this relative record number just prior to the code 3 call by using a code 5 ISAM statement. (A code 5 call returns the relative record number of the next free data record.)

CODE 4 - Delete the specified symbolic key from the index file selected by #file-channel. This function returns the corresponding relative record number so that you can use a code 6 ISAM statement to delete the data record and return it to the free list. If the function cannot find the symbolic key in the index file, it returns a "?Record not found" error (33).

CODE 5 - Finds the next available data record on the free list. (The free list is a linked list that keeps track of all available records in the data file. ISMBLD initially builds the free list.) Returns the relative record number of that record so that you can use a code 3 ISAM statement to add a symbolic key/relative record number pair to the index file. If no more data records are free in the data file, the function returns a "?Data file full" error. A code 5 ISAM statement does not modify the index file; it simply locates the next free record in the data file. The function ignores the symbolic key in the ISAM statement. The #file-channel in the code 5 ISAM statement must be the file channel assigned to the primary index file.

CODE 6 - Returns to the free list the data record specified by the relative record number in the OPEN statement. Does not modify the index file. The #file-channel in the code 6 ISAM statement must be the file channel assigned to the primary index file. A code 6 call ignores the symbolic key in the ISAM statement.

CODE 7 - Accesses the index file selected by #file-channel and finds the next symbolic key. Returns the relative record number of the data file record associated with that symbolic key in preparation for a READ or a WRITE to the data file. If this is the first access to the file after the OPEN statement, it finds the first symbolic key in the index file. If this function follows a previous code 1 statement, the function finds the next symbolic key after the code 1 symbolic key. If there are no more keys in the index file, the function returns an end-of-index-file error (38): make no further accesses to the data file until you make another ISAM call that returns a valid relative record number.

NOTE: This code performs exactly the same function as code 2 above, except that it returns the symbolic key as well as the relative record number.

It is very important that the symbolic key variable that appears in your code 7 ISAM statement be the same size as or larger than the key defined in the ISAM index file. If the variable is smaller than the key, data following the symbolic key in memory will be overwritten, damaging your running program.

10.3 READING AND WRITING DATA IN AN ISAM DATA FILE

ISAM statements do not access data records, but instead return their relative record numbers. To actually read or write data records, you must use the AlphaBASIC READ, READL, WRITE, and WRITEL commands. When you read or write data in a specific ISAM data file, AlphaBASIC selects the record to be accessed by referring to the relative record number variable in the OPEN statement for that file.

```
READ #file-channel, variable1, variable2,... variableN
READL #file-channel, variable1, variable2,... variableN
WRITE #file-channel, variable1, variable2,... variableN
WRITEL #file-channel, variable1, variable2,... variableN
```

The #file-channel in the READ, READL, WRITE, or WRITEL statements MUST be the file channel that appears in the OPEN statement for the primary index file you want to access. The relative record number variable in the OPEN statement must contain a valid relative record number or an error will result.

READL and WRITEL only lock the record if LOKSER is installed on your system; on a non-LOKSER system, they perform the same actions as READ and WRITE. However, even if your system does not use LOKSER, we recommend that you use READL and WRITEL, since this will promote program standardization and will make it simple for you to transfer your programs to a LOKSER system at a later date.

READL locks the current record until you use the UNLOKR statement (see Section 10.5, below) or until you use a WRITE statement to output the record.

WRITEL locks the current record until it finishes outputting the record.

10.4 CLOSING FILES

To ensure that ISAM has rewritten all data records to the data file and that it has properly updated all links in the index file, it is VERY important that you close all index files (primary and secondary) via the normal CLOSE statement. Failing to close the file when you are through with it may destroy the linking structure of the indexed sequential file. The CLOSE statement takes the form:

```
CLOSE #file-channel
```

where #file-channel is the file channel assigned to the file you want to close. For example:

```
CLOSE #200
```

where file channel #200 was assigned to an indexed sequential file by a previous OPEN statement. Remember to close both primary and secondary index files. NOTE: The order in which you close the ISAM files makes no difference; however, remember that you cannot access a secondary index file if you have already closed the primary index file/data file.

10.5 LOCKING AND UNLOCKING STATEMENTS

If LOKSER is installed on your system, the OPEN statement that opens an ISAM file automatically locks the ISAM file for Shared or Exclusive use (depending on the type of file mode you specify). This section discusses the various options open to you if you want to explicitly lock and unlock files and records.

When actually accessing the data in a file opened for Shared use, you can lock the data records using the READL or WRITEL statements. A WRITE operation automatically unlocks the record locked by the READL statement. The WRITEL statement unlocks the record it is outputting when it is finished.

You can explicitly lock the primary index file by using the LOCK statement:

```
LOCK #File-channel
```

If you locate a record using the primary index file, LOKSER automatically locks the primary index file for you. However, if you locate a record using the secondary index file, you must use the LOCK statement to explicitly lock the primary index file. Whether or not you have the WAIT'FILE clause set in your ISAM file OPEN statement, the LOCK statement will always cause your program to wait for the primary index file if it is already in use.

The statement:

```
UNLOKR #File-channel
```

unlocks the primary index file and any record currently locked via a READL statement.

10.6 ERROR PROCESSING

Any ISAM operation can result in some kind of error. If the error is a system error (for example, the disk is not mounted), AlphaBASIC interrupts your program and aborts to the monitor. (Or, if error trapping is enabled, AlphaBASIC transfers control to your error handling routine.) To check for hard system errors, use the ERR(X) function.

Special ISAM errors can also occur as a result of an ISAM operation. These errors do not generate an error message or result in an error trap. It is therefore very important that your program check for these errors after every ISAM statement; otherwise, you have no way of knowing whether or not the ISAM function was performed successfully. To do so, use the ERF(X) function, where X is the file channel number used by the preceding ISAM statement. (The ERF(X) function operates in much the same way as the EOF(X) function.)

If ERF(X) returns a zero, the preceding ISAM statement was successful. If ERF(X) returns a nonzero value, then an error was detected. If an error occurred, your program should correct the problem before going on to access the file. The nonzero value returned tells you which error occurred. For example:

```
! If a "Record not found" error (#33), go to routine that
!    asks for new key.
100 IF ERF(2)=33 &
        THEN PRINT "RECORD NOT FOUND" : GOTO PROMPT
```

The current ISAM error codes are:

32 - Illegal ISAM statement code.
33 - Record not found in index file search.
34 - Duplicate key found in index file during attempted key addition.
35 - Link structure is smashed and must be re-created.
36 - Index file is full.
37 - Data file is full (free list is empty).
38 - End of file during sequential key read.
39 - Illegal ISAM sequence

The last message occurs if LOKSER is installed on your system and you do not perform the ISAM file access procedures in the order given in Section 10.7.

REMEMBER: Always check after performing an ISAM function to see if an error occurred. If you do detect an error, your program must take corrective action before continuing on.

For complete examples of performing error checking and handling from within an AlphaBASIC program that uses ISAM, refer to the program examples in Chapter 6.

10.7 USING THE ISAM FUNCTIONS WITHIN AN ALPHABASIC PROGRAM

Below are some examples of the ways you can combine the ISAM statements and other AlphaBASIC commands to access and use indexed sequential files. For a look at a sample ISAM program, turn to Section 10.8. Also see the program examples in Chapter 6.

10.7.1 Adding Data to an Indexed Sequential File

At the time that you use ISMBLD to create an indexed sequential file, you have the option of loading data into the ISAM data and primary index file from an ordinary sequential data file. Your AlphaBASIC programs may also add data to the indexed sequential file by using code 5 and code 3 ISAM statements. For each new data record to be added:

1. Open the indexed sequential file with an OPEN statement, specifying INDEXED mode. For example:

```
OPEN #100,"PHONES",INDEXED,RECSIZE,RELKEY
```

   Remember to open any secondary index files that you might want to use via separate OPEN statements on different file channels. If LOKSER is installed on your system, the OPEN statement above will open the ISAM file for Shared use.

2. Use a code 1 statement to see if the index entry you want to add already exists. For example:

```
ISAM #100, 1, NAME
```

Check to see if an error was returned. For example:

```
IF ERF(100) = 0 &
  THEN PRINT "Duplicate Name." : UNLOKR #100 : GOTO GET'NAME
```

(If no error occurred, then the index entry already exists and you can't add it.) If you are using secondary index files, also check to see that the secondary index entries don't already exist.

3. Retrieve the next free data record (a code 5 ISAM statement). For example:

```
ISAM #100, 5, DUMMY
```

Check to make sure that an error (e.g., 37 - "?Data file is full (free list is empty)" did not occur. For example:

```
IF ERF(100) <> 0 THEN GOTO ISAM'ERROR
```

4. If no error occurred, the record number of the next free record is in the relative record number variable defined by the OPEN statement for the indexed sequential file. Now you can write the data into the record by using a WRITEL statement. For example:

```
WRITEL #100,INFO
```

(Note: Even if LOKSER is not installed on your system, proper coding techniques require that you use WRITEL. Of course, since LOKSER is not installed, WRITEL will not lock the record.)

5. Now you must add the symbolic keys for that data record to the index files, using a code 3 statement. (Those symbolic keys will then link to that data record.) Be sure to check for an ISAM error after each addition. For example:

```
ISAM #100, 3, NAME
IF ERF(100) <> 0 THEN GOTO ISAM'ERROR
```

6. After adding all data records, close the ISAM files. For example:

```
CLOSE #100      ! Close primary index file/data file
CLOSE #200      ! Close secondary index file
```

7. Your error trapping routine should contain the UNLOKR statement to unlock the primary index file and release the current record if an error occurs. For example:

```
ISAM'ERROR:
     UNLOKR #100
```

10.7.2 Reading Data Records in Symbolic Key Order

ISAM stores symbolic keys in the index file in ASCII collating sequence. To retrieve records in the order in which their keys appear in an index file:

1. Open the indexed sequential file with an OPEN statement. (If you also wish to open one or more secondary index files that cross-index to the primary index file, use one OPEN statement for each secondary index file.)

2. Execute a code 2 ISAM statement to find the next symbolic key.

3. Check to make sure that the ISAM statement didn't return an error. For example:

```
IF ERF(100) = 38 THEN PRINT "End of file." : GOTO PROMPT
IF ERF(100) <> 0 THEN GOTO ISAM'ERROR
```

4. The proper record number is now in the relative record number defined by the OPEN statement for the file, so you can use a READ statement to read in the data. For example:

```
READ #100, INFO
```

   Remember that the READ statement must include the file channel assigned to the primary index file even if the code 2 ISAM statement included a symbolic key contained in a secondary index file; this is because the data you want to read is in the data file. Also, you must use the LOCK statement to lock the primary index file if you located the record via a key in the secondary index file.

5. Check for an end-of-file error by using the ERF(X) function.

6. Repeat these procedures to step through the data records in the order of the symbolic keys in the index files until you reach the end of the file, or until you have accessed all the records you need. Be sure to check for an ISAM error after each access.

7. Close all files when you are done.

10.7.3 Reading Data Records Randomly by Symbolic Key

1. Open the indexed sequential file with an OPEN statement. You must include one OPEN statement for the data file/primary index file. You must also include one OPEN statement for each secondary index file you want to access.

2. Locate each data record by using a code 1 ISAM statement. The statement must contain the symbolic key associated with the record for which you are searching and the file channel associated with the index file containing the symbolic key.

3. Check for a "record not found" error; this indicates that the symbolic key was not located in the specified index file.

4. If the record was found, use a READ statement to read in the data record. The READ statement includes the file channel associated with the data file/primary index file, even if the symbolic key used belonged to a secondary index file. If the symbolic key belonged to a secondary index file, remember that you must lock the primary index file before using the READ statement.

5. Repeat steps 2 through 4 for each record you want to find.

6. Close all files.

10.7.4 Updating Data Records

1. Open the indexed sequential file with an OPEN statement.

2. Locate the data record you want to update via one of the methods above (i.e., by using a code 1 or code 2 ISAM statement).

3. Check to make sure that the record was found. (Use the ERF function.)

4. Read the data record from the ISAM data file, specifying the file channel of the primary index file.

```
READL #100, DATA'MAP
```

(NOTE: Even if LOKSER is not installed on your system, proper coding techniques require that you use READL. Of course, if LOKSER is not installed, a READL will not lock the record and primary index file.)

5. Make a copy of the relative key. For example:

```
OLD'KEY = NEW'KEY
```

6. Get the new data to put into the record from the user of the program.

7. Check the key to see if it changed:

```
IF OLD'KEY = NEW'KEY THEN GOTO SAME'RECORD
```

8. If the key did not change, use the WRITE statement to update (and unlock) the data record. (The WRITE statement includes the file channel associated with the data file/primary index file, even if the symbolic key used to find the record belonged to a secondary index file.)

9. If the key did change, you will need to change the index files.

   First, make sure the key is unique:

```
ISAM #100, 1, NEW'KEY
IF ERF(100) = 0 THEN GOTO ISAM'ERROR
```

   Next, you must delete the old record (as discussed in Section 10.7.5) and add the new record (Section 10.7.1).

10. Close all files.

11. Your error trapping routine should contain the UNLOKR statement to unlock the primary index file and release the current record if an error occurs.

10.7.5 Deleting a Data Record

Deleting a data record from an indexed sequential file entails not only deleting the record itself but also deleting all symbolic keys associated with that data record from all index files.

1. Open the primary index file and all secondary index files needed.

2. Locate the data record via one of the symbolic keys (a code 1 ISAM statement).

3. Check to make sure that the statement executed without error. For example:

```
IF ERF(200) = 33 THEN PRINT "Record not found." : GOTO PROMPT
IF ERF(200) <> 0 THEN GOTO ISAM'ERROR
```

4. Read the data record with a READL statement (whose #file-channel is the file channel number associated with the primary index file). (Use READL even if LOKSER is not installed on your system.) If you located the data record using a symbolic key in a secondary index file, you must first LOCK the primary index file before doing the READL.

5. Extract each symbolic key from that data record. Use each symbolic key to delete each key from its associated index file with code 4 ISAM statements.

6. After all symbolic keys have been deleted from all index files, delete the record itself via a code 6 ISAM statement.

7. Close all files.

8. Your error trapping routine should contain the UNLOKR statement to unlock the primary index file and release the current record if an error occurs.

10.8 SAMPLE ALPHABASIC ISAM PROGRAM

The sample program in the section below will make clearer the use of the commands discussed above. For more information on using ISAM from within an AlphaBASIC program, consult the manual AlphaBASIC User's Manual.

Before we can begin to use ISAM, we must load it into memory if it is not already resident in system memory:

```
.LOAD SYS:ISAM.SYS RET
```

Before we run the sample AlphaBASIC program, we first use the program ISMBLD to build the ISAM files LABELS.IDA (the data file), LABELS.IDX (the primary index file), and HASH.IDX (the secondary index file). Note that we build an empty file (i.e., we type a RETURN after the "Load from file:" prompt). We use the AlphaBASIC program below to place data into the file.

10.8.1 Sample ISMBLD Use on an AMOS System

```
.ISMBLD LABELS RET
Size of key: 25 RET
Position of key: 1 RET
Size of data record: 67 RET
Number of records to allocate: 50 RET
Entries per index block: 10 RET
Empty index blocks to allocate: 20 RET
Primary Directory: Y RET
Data file device: RET

Load from file: RET

.ISMBLD HASH RET
Size of key: 10 RET
Position of key: 58 RET
Size of data record: 67 RET
Number of records to allocate: 50 RET
Entries per index block: 10 RET
Empty index blocks to allocate: 20 RET
Primary Directory? N RET

Secondary index to file: LABELS RET
End of primary file
No records loaded
```

10.8.2 Sample ISMBLD Use on an AMOS/L System

```
.ISMBLD LABELS(RET)
Size of key: 25(RET)
Position of key: 1(RET)
Size of data record: 67(RET)
Number of records to allocate: 50(RET)
        6 index blocks will be allocated
Empty index blocks to allocate: 6(RET)
Primary Directory: Y(RET)
Data file device:(RET)

Load from file:(RET)

.ISMBLD HASH(RET)
Size of key: 10(RET)
Position of key: 58(RET)
Size of data record: 67(RET)
Number of records to allocate: 50(RET)
        4 index blocks will be allocated
Empty index blocks to allocate: 4(RET)
Primary Directory? N(RET)

Secondary index to file: LABELS(RET)
End of primary file
No records loaded
```

10.8.3 Sample AlphaBASIC Program

Once you have performed the procedures above, you can run the .RUN version of the program below:

```
.RUN MAIL
```

THE SAMPLE ALPHABASIC PROGRAM

```
10 ! ISAM Sample Program.
20 !
30 ! This program is a simple example of how to handle ISAM files, both
40 ! primary and secondary.  It simulates a very simple-minded mailing
50 ! list program, with the addresses keyed by both name and user
60 ! defined hash code.
70 !
90 !
110  MAP1 LABEL                                  ! Define the Mailing List file
120     MAP2 NAME,S,25                           !   record.
130     MAP2 ADDRESS,S,25
140     MAP2 STATE,S,2
150     MAP2 ZIP,S,5
160     MAP2 HASH,S,10
170  MAP1 FIELDS
180     MAP2 NEW'ADDRESS,S,25
190     MAP2 NEW'STATE,S,2
200     MAP2 NEW'ZIP,S,5
210  MAP1 RECSIZE,F,6,67                                 ! Size of data record.
220                                                  ! Open the primary and secondary
230                                                  !   index files.
240     OPEN #100, "LABELS", INDEXED, RECSIZE, RELKEY1
250     OPEN #200, "HASH", INDEXED, RECSIZE, RELKEY1

260  PROMPT:
270     PRINT
280     INPUT "ENTER FUNCTION &
          (1=ADD,2=DELETE,3=INQUIRE,4=PRINT,5=CHANGE,99=END): "; FUNCTION
290     ON FUNCTION GOTO ADD'RECORD,DELETE'RECORD,INQUIRE'RECORD,&
                PRINT'LABELS,CHANGE'RECORD
300     IF FUNCTION=99 &
                THEN GOTO END'IT
310     GOTO PROMPT

320  ADD'RECORD:
330     INPUT "ENTER NAME: "; NAME
340     INPUT "ENTER ADDRESS: "; ADDRESS
350     INPUT "ENTER STATE: "; STATE
360     INPUT "ENTER ZIP: "; ZIP
370     INPUT "ENTER HASH: "; HASH
380     NAME = NAME + SPACE(25-LEN(NAME))        ! Add trailing blanks to keys
390     HASH = HASH + SPACE(10-LEN(HASH))
```

```
400     ISAM #100, 1, NAME                      ! Look up name to verify that
410                                             !   it is not a duplicate.
420     IF ERF(100) = 0 &
                THEN PRINT "DUPLICATE NAME" : UNLOKR #100 : GOTO ADD'RECORD
430                                             ! If ERF(100)=0, then ISAM
440                                             !   found the key in data file
450     ISAM #200, 1, HASH                      ! Verify that hash is not
460                                             !   duplicate.
470     IF ERF(200) = 0 &
                THEN PRINT "DUPLICATE HASH" : UNLOKR #100 : GOTO ADD'RECORD
480     ISAM #100, 5, NAME                      ! Get free data record from
490                                             !   primary index
500     IF ERF(100) <> 0 &
                THEN GOTO ISAM'ERROR
510     WRITEL #100, LABEL                      ! Write out record.
520     ISAM #200, 3, HASH                      ! Add key to secondary index
530     IF ERF(200) <> 0 &
                THEN GOTO ISAM'ERROR
540     ISAM #100, 3, NAME                      ! Add key to primary index.
550     IF ERF(100) <> 0 &
                THEN GOTO ISAM'ERROR
560     GOTO PROMPT

570  DELETE'RECORD:
580     INPUT "ENTER NAME: "; NAME
590     NAME = NAME + SPACE(25-LEN(NAME))
610     ISAM #100, 1, NAME                      ! Verify that the key exists
620     IF ERF(100) = 33 &
                THEN PRINT "RECORD NOT FOUND" : UNLOKR #100 : GOTO PROMPT
630     IF ERF(100) <> 0 &
                THEN GOTO ISAM'ERROR
640     READL #100, LABEL
650     ISAM #200, 4, HASH                      ! Delete key from secondary
660     IF ERF(200) <> 0 &
                THEN GOTO ISAM'ERROR
670     ISAM #100, 4, NAME                      ! Delete key from primary
680     IF ERF(100) <> 0 &
                THEN GOTO ISAM'ERROR
700     ISAM #100, 6, NAME                      ! Delete data record from
710                                             !   data file.
720     IF ERF(100) <> 0 &
                THEN GOTO ISAM'ERROR
730     GOTO PROMPT

740  INQUIRE'RECORD:
750     INPUT "BY NAME (1) OR HASH (2): "; FUNCTION
760     IF FUNCTION = 2 &
                THEN GOTO BY'HASH
770     INPUT "NAME: "; NAME
780     NAME = NAME + SPACE(25-LEN(NAME))
790     ISAM #100, 1, NAME                      ! Locate the record
```

```
800     IF ERF(100) = 33 &
                THEN PRINT "RECORD NOT FOUND" : UNLOKR #100 : GOTO PROMPT
810     IF ERF(100) <> 0 &
                THEN GOTO ISAM'ERROR

820  READ'RECORD:
830     READ #100, LABEL                          ! Read the record
840     PRINT NAME, HASH
850     PRINT ADDRESS, STATE, ZIP
860     GOTO PROMPT

870  BY'HASH:                                     ! Locate record by hash
880     INPUT "HASH: "; HASH
890     HASH = HASH + SPACE(10-LEN(HASH))
900     LOCK #100                                 ! Lock primary index file
910     ISAM #200, 1, HASH
920     IF ERF(200) = 33 &
                THEN PRINT "RECORD NOT FOUND" : UNLOKR #100 : GOTO PROMPT
930     IF ERF(200) <> 0 &
                THEN GOTO ISAM'ERROR
940     GOTO READ'RECORD

950   PRINT'LABELS:
960     NAME = SPACE(25)                          ! Read null key to get to
970     ISAM #100, 1, NAME                        !   front of file.
975   UNLOKR #100                                 ! Release the index.
980  LOOP:                                        ! Loop through file doing
990                                               !   sequential reads until
1000                                              !   we hit end of file.
1010    ISAM #100, 2, NAME
1020    IF ERF(100) = 38 &
                THEN GOTO PROMPT                  ! We hit end-of-file.
1030    IF ERF(100) <> 0 &
                THEN GOTO ISAM'ERROR
1040    READ #100, LABEL
1050    PRINT
1060    PRINT NAME, HASH
1070    PRINT ADDRESS, STATE, ZIP
1080    GOTO LOOP

1090 CHANGE'RECORD:
1100    INPUT "BY NAME (1) OR HASH (2): "; FUNCTION
1110    IF FUNCTION = 2 &
                THEN GOTO BY'HASH'CHANGE
1120    INPUT "NAME: "; NAME
1130    NAME = NAME + SPACE(25-LEN(NAME))
1140    ISAM #100, 1, NAME                        ! Locate the record
1150    IF ERF(100) = 33 &
                THEN PRINT "RECORD NOT FOUND" : UNLOKR #100 : GOTO PROMPT
1160    IF ERF(100) <> 0 &
                THEN GOTO ISAM'ERROR
```

```
1170 READ'RECORD'CHANGE:
1180    READL #100, LABEL
1190    PRINT NAME, HASH
1200    PRINT "ADDRESS = "+ADDRESS
1210    INPUT "NEW ADDRESS: "; NEW'ADDRESS
1220    IF LEN(NEW'ADDRESS) > 0 &
                THEN ADDRESS = NEW'ADDRESS
1230    PRINT "STATE ="+STATE
1240    INPUT "NEW STATE ="; NEW'STATE
1250    IF LEN(NEW'STATE) > 0 &
                THEN STATE = NEW'STATE
1260    PRINT "ZIP = "+ZIP
1270    INPUT "NEW ZIP ="; NEW'ZIP
1280    IF LEN(NEW'ZIP) > 0 &
                THEN ZIP=NEW'ZIP
1290    WRITE #100, LABEL
1300    GOTO PROMPT

1310 BY'HASH'CHANGE:
1320    INPUT "HASH: "; HASH
1330    HASH = HASH + SPACE(10-LEN(HASH))
1340    LOCK #100
1350    ISAM #200, 1, HASH
1360    IF ERF(200) = 33 &
                THEN PRINT "RECORD NOT FOUND" : UNLOKR #100 : GOTO PROMPT
1370    IF ERF(200) <> 0 &
                THEN GOTO ISAM'ERROR
1380    GOTO READ'RECORD'CHANGE
1390

1400 END'IT:                                    ! Be sure to close files
1410    CLOSE #100                              !   before we exit!
1420    CLOSE #200
1430    END

1440 ISAM'ERROR:                                ! ERF(X) returned ISAM error
1450    PRINT "?FATAL ISAM ERROR"               ! other than RECORD NOT FOUND.
1460    PRINT "ISAM ERROR "+ERF(100)+"; SECONDARY: "+ERF(200)
1470    UNLOKR #100
1480    END
```

Index

Adding data records 10-9
Adding symbolic keys 7-10, 8-8, 10-5
AMOS monitor calls 7-1
  COPY 7-2
  EXIT 7-3
  FETCH 7-1
  GETMEM 7-1, 7-3
AMOS/L monitor calls 8-1
  COPY 8-1
  EXIT 8-3
  FETCH 8-1
  GETMEM 8-1, 8-3

BASIC
  Adding data records 10-9
  Closing files 10-7
  Deleting data records 10-13
  ERF(X) 10-4, 10-8
  Error processing 10-8
  ISAM codes 10-4
  ISAM error codes 10-9
  ISAM statement 10-3
  OPEN statement 10-1
  Opening an ISAM file 10-1
  READ statement 10-4, 10-6
  Reading data records . . . . . 10-11 to 10-12
  READL statement 10-6
  Sample ISAM program 10-15
  Updating data records 10-12
  WRITE statement 10-4, 10-6
  WRITEL statement 10-6
Block split 3-5

CLOSE 10-7
Closing ISAM files 2-6, 7-5, 8-5, 10-7
Code 10-4
Completion codes 7-2, 8-2, 9-2
  IS.EOF 7-11, 8-9
  IS.SUC 7-2, 8-2
Condition codes 7-2, 8-2
Condition flags 7-2, 8-2
Contiguous file 2-4

Counted update mode 2-5, 7-4, 8-4, 10-2
Creating ISAM files 3-1, 10-1

Data file . . . . . 2-2 to 2-4
Deleting data records 7-6, 8-6, 10-5, 10-13
Deleting symbolic keys 7-10, 8-8, 10-5
Directory Rock 2-5
Displaying the data file 4-2
Displaying the index file 4-2

ERF 6-11
ERF function 10-4
Error processing 10-8
Exclusive open mode . . . . . 2-5 to 2-6, 7-4, 8-4, 10-2

File channel 10-2, 10-4
File interlocking . . . . . 2-5 to 2-6
File locking 6-8
File pair number . . . . . 7-3 to 7-4, 8-3 to 8-4
File parameters 3-2
Filespec 10-2
Finalizing ISAM processing . . . 7-3, 8-3
Finding data records 7-9, 8-7, 10-4
Finding free data records 7-5, 8-5, 10-5
Finding symbolic keys 10-4
Finding the next key . . . . . 7-11, 8-9, 10-5 to 10-6
Flags 9-2
FLOCK . . . . . 6-2 to 6-3, 6-5

Index file . . . . . 2-2 to 2-3
Index levels 2-4
INDEXED 10-2
Indexed sequential file 2-3
  Data file 2-3
  Index file 2-3
INDEXED'EXCLUSIVE 2-6
Indicators 7-2, 8-2
Initializing ISAM 7-2, 8-2
ISAM 1-1
ISAM access modes 2-5
ISAM calls 7-1, 8-1, 9-1
  .ICLOS 7-2, 7-5, 8-2, 8-5
  .IDELK 7-10, 8-8
  .IDLFR 7-6, 8-6
  .IFIN 7-3, 8-3
  .IGTFR 7-5, 8-5
  .INIT . . . . . 7-2 to 7-3, 8-2 to 8-3
  .IOPNR 7-4, 8-4
  .IREDR 7-9, 7-11, 8-7, 8-9
  .IRLRD 7-7, 8-6
  .IRLWT 7-8, 8-7
  .IWRTR 7-10, 8-8

.SREDR 7-11, 8-9
ISAM codes 10-4
ISAM error codes 10-9
ISAM file extensions
.IDA 2-4
.IDX 2-5
.IDY 2-6
ISAM statement 10-3
ISMBLD . . . . . 2-3, 3-1, 10-1, 10-15 to 10-16
ISMDMP 4-1
ISUSYM.M68 8-1
ISUSYM.MAC 7-1

Loading ISAM files 3-1
LOCK 6-9, 6-13, 10-7
LOKSER 6-3, 6-7, 10-2

Memory allocation routine 7-3, 8-3
Memory deallocation routine . . . 7-3, 8-3
Memory requirements 7-1, 8-1
Mode 10-2

Opening ISAM files 7-4, 8-4, 10-1

Primary index file 2-3, 2-5

READ 10-6
Reading data records . . . . . 7-7, 8-6, 10-11 to 10-12
READL 10-6
Record size 10-2
Relative key 2-2
Relative record number 2-2, 10-2
Return codes 7-1, 8-1

Sample BASIC ISAM program 10-15
Sample ISMBLD dialog 10-15
Secondary index file 2-3, 2-5
Suppressing Exclusive Open mode . 2-7, 3-5, 4-2
Symbolic key 10-4

Table offsets 7-1, 8-1

Unloading ISAM files 4-1
UNLOKR . . . . . 6-9 to 6-10, 10-8
Updating data records 7-8, 8-7, 10-12

WAIT'FILE 10-2
WAIT'RECORD 10-2
WRITE 10-6
WRITEL 10-6
Writing data records 7-8, 8-7

XLOCK . . . . . 6-2 to 6-4